



MySQL

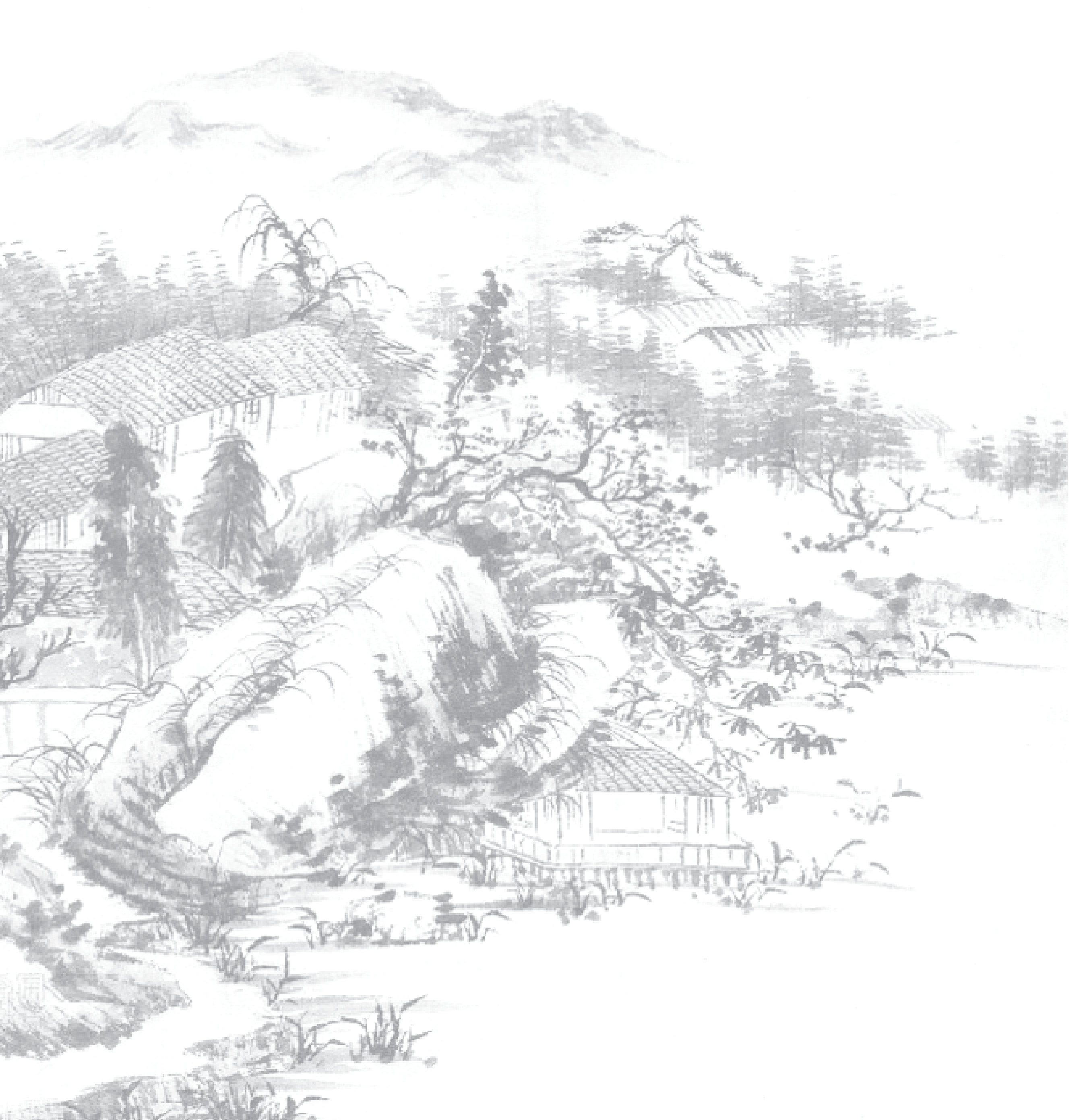
从入门到精通

6小时语音视频讲解

明日科技 编著



☑ 实例资源库 ☑ 模块资源库 ☑ 项目资源库
☑ 面试资源库 ☑ 测试题库系统 ☑ PPT电子课件



循序渐进，实战讲述

基础知识 ➪ 核心技术 ➪ 高级应用 ➪ 项目实战
219个学习实例，46个练习实例，2个项目案例

海量资源，可查可练

除本书配套的6小时视频讲解外，根据学习顺序，光盘还额外配备如下海量开发资源库：

实例资源库（808个实例）➪ 模块资源库（15个典型模块）➪ 项目资源库（15个项目案例）➪ 测试题库系统（626道测试题）➪ 面试资源库（342个面试真题）

在线解答，高效学习

QQ: 400 675 1066（可容纳10万人在线）

清华大学出版社

软件开发视频大讲堂

MySQL 从入门到精通

明日科技　编著

清华大学出版社
北　京

内 容 简 介

《MySQL 从入门到精通》从初学者角度出发，通过通俗易懂的语言以及丰富多彩的实例，详细介绍了 MySQL 开发应该掌握的各方面技术。全书共分为 4 篇 23 章，包括数据库基础，初识 MySQL，使用 MySQL 图形化管理工具，数据库操作，存储引擎及数据类型，操作数据表，MySQL 基础，表数据的增、删、改操作，数据查询，常用函数，索引，视图，数据完整性约束，存储过程与存储函数，触发器，事务的应用，事件，备份与恢复，MySQL 性能优化，权限管理及安全控制，PHP 管理 MySQL 数据库中的数据，Apache+PHP+MySQL 实现网上社区，Struts 2+Spring+Hibernate+MySQL 实现网络商城等内容。所有知识都结合具体实例进行介绍，涉及的程序代码也给出了详细的注释，可以使读者轻松领会 MySQL 的精髓，快速提高开发技能。

另外，本书除了纸质内容之外，配套光盘中还给出了海量开发资源库，主要内容如下：

- ☑ 语音视频讲解：总时长 6 小时，共 68 段
- ☑ 实例资源库：808 个实例及源码详细分析
- ☑ 模块资源库：15 个经典模块开发过程完整展现
- ☑ 项目案例资源库：15 个企业项目开发过程完整展现
- ☑ 测试题库系统：626 道能力测试题目
- ☑ 面试资源库：342 个企业面试真题
- ☑ PPT 电子教案

本书内容详尽，实例丰富，非常适合作为编程初学者的学习用书，也适合作为开发人员的查阅、参考资料。

图书在版编目（CIP）数据

MySQL 从入门到精通/明日科技编著. —北京：清华大学出版社，2017
（软件开发视频大讲堂）
ISBN 978-7-302-45799-2

I. ①M…　II. ①明…　III. ①SQL 语言-程序设计　IV. ①TP311.132.3

中国版本图书馆 CIP 数据核字（2016）第 290847 号

责任编辑： 赵洛育
封面设计： 刘洪利
版式设计： 李会影
责任校对： 何士如
责任印制：

出版发行： 清华大学出版社
网　　址： http://www.tup.com.cn，http://www.wqbook.com
地　　址： 北京清华大学学研大厦 A 座　　**邮　　编：** 100084
社 总 机： 010-62770175　　**邮　　购：** 010-62786544
投稿与读者服务： 010-62776969，c-service@tup.tsinghua.edu.cn
质量反馈： 010-62772015，zhiliang@tup.tsinghua.edu.cn
印 刷 者：
装 订 者：
经　　销： 全国新华书店
开　　本： 203mm×260mm　**印　　张：** 32.5　**字　　数：** 889 千字
（附 DVD 视频光盘 1 张）
版　　次： 2017 年 9 月第 1 版　**印　　次：** 2017 年 9 月第 1 次印刷
印　　数： 1～5000
定　　价： 79.80 元

产品编号：058862-01

如何使用本书开发资源库

在学习《MySQL 从入门到精通》一书时，随书附配光盘提供了“PHP 开发资源库”系统，可以帮助读者快速提升编程水平和解决实际问题的能力。《MySQL 从入门到精通》和 PHP 开发资源库配合学习流程如图 1 所示。

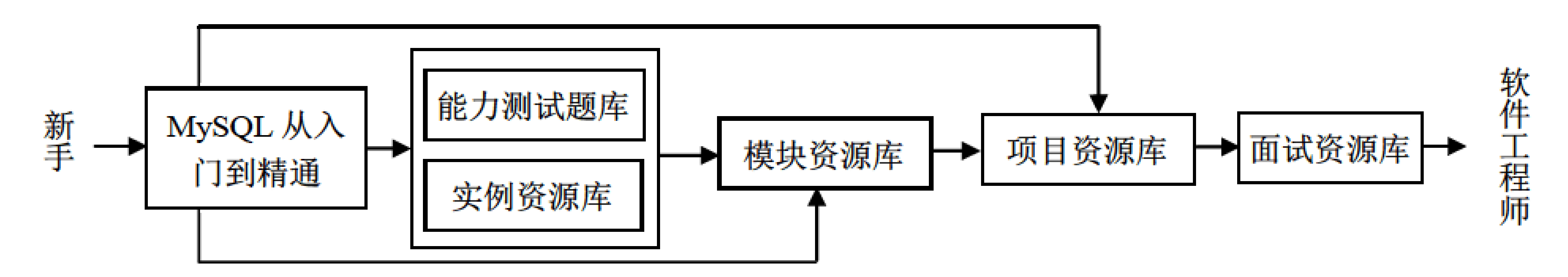


图 1　图书与开发资源库配合学习流程图

打开光盘的“开发资源库”文件夹，运行 PHP 开发资源库.exe 程序，即可进入“PHP 开发资源库”系统，主界面如图 2 所示。

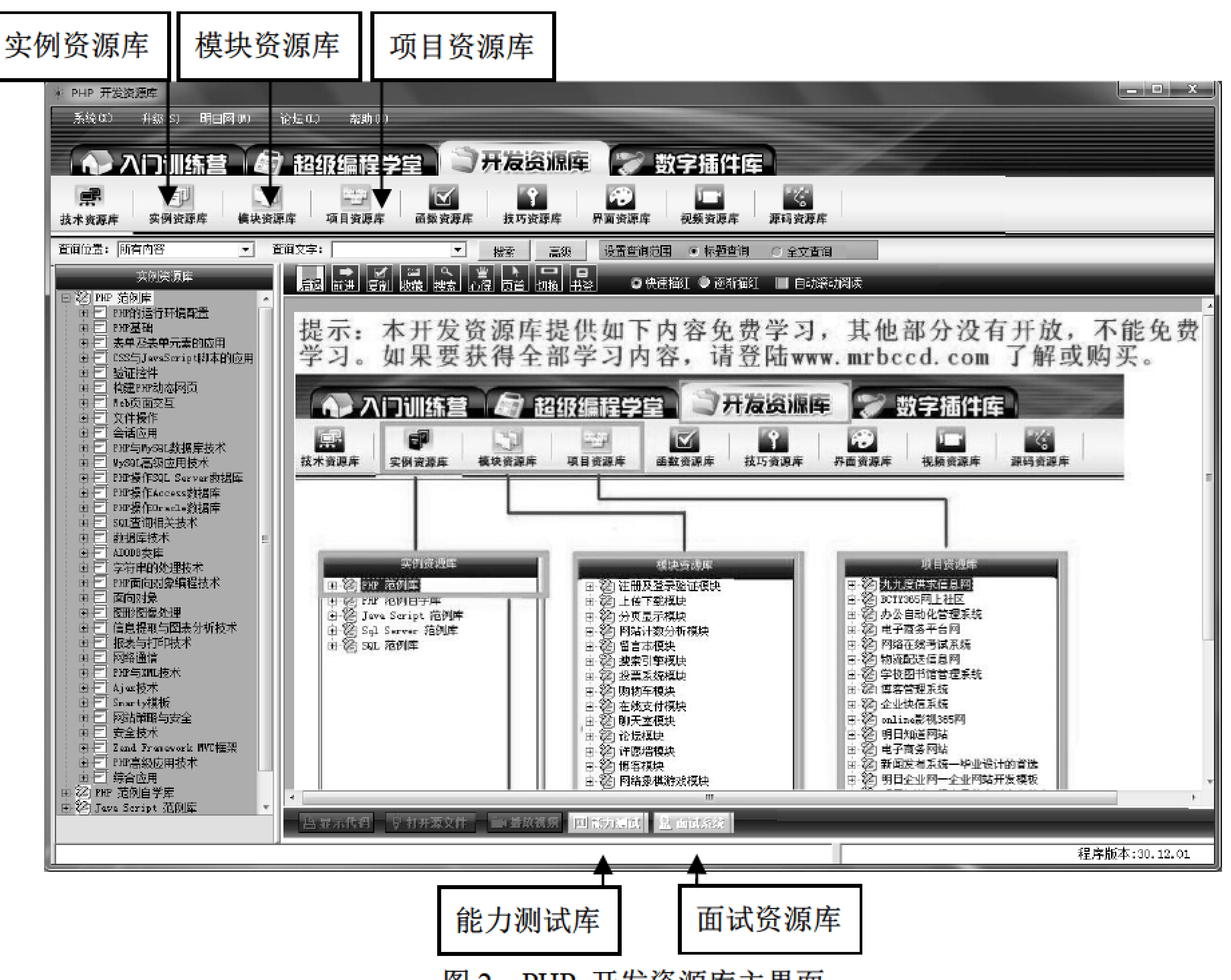


图 2　PHP 开发资源库主界面

在学习某一章节时，可以配合实例资源库的相应章节，利用实例资源库提供的大量热点实例和关

键实例巩固所学编程技能，提高编程兴趣和自信心；也可以配合能力测试题库的对应章节进行测试，检验学习成果。具体流程如图 3 所示。

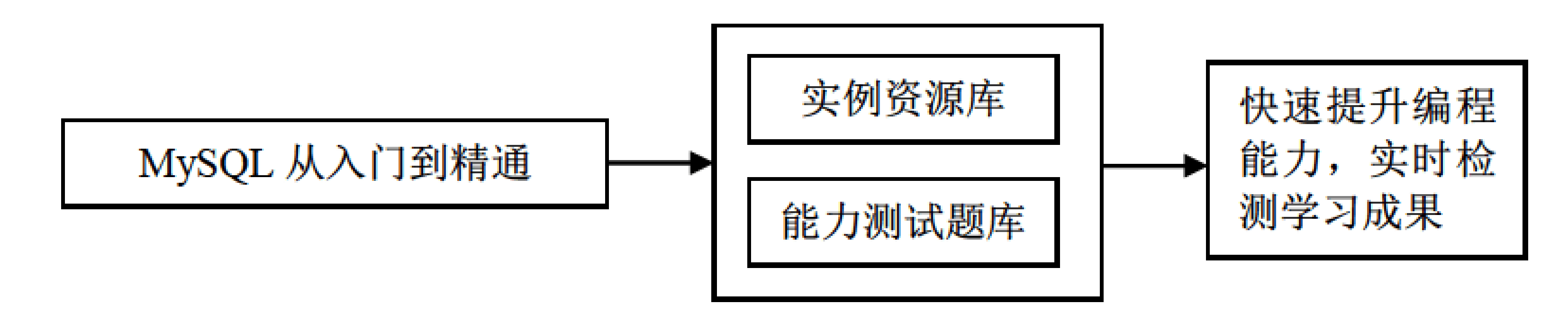


图 3　使用实例资源库和能力测试题库

对于数学逻辑能力和英语基础较为薄弱的读者，或者想了解个人数学逻辑思维能力和编程英语基础的用户，本书提供了数学及逻辑思维能力测试和编程英语能力测试供练习和测试，如图 4 所示。

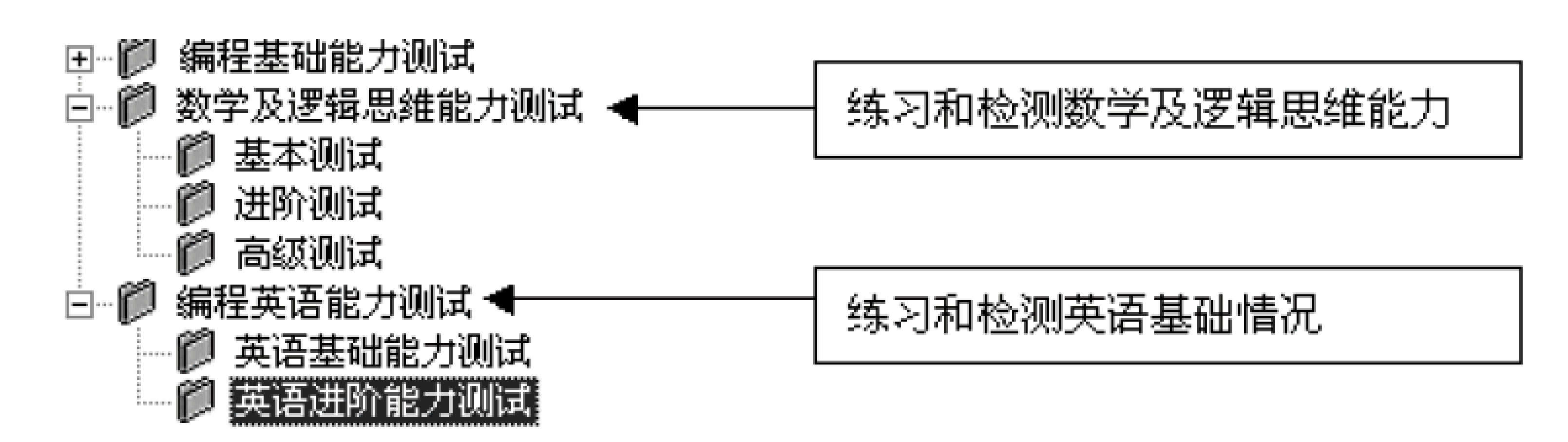


图 4　数学及逻辑思维能力测试和编程英语能力测试目录

当本书学习完成时，可以配合模块资源库和项目资源库的 30 个模块和项目，全面提升个人综合编程技能和解决实际开发问题的能力，为成为 PHP 软件开发工程师打下坚实基础。具体模块和项目目录如图 5 所示。

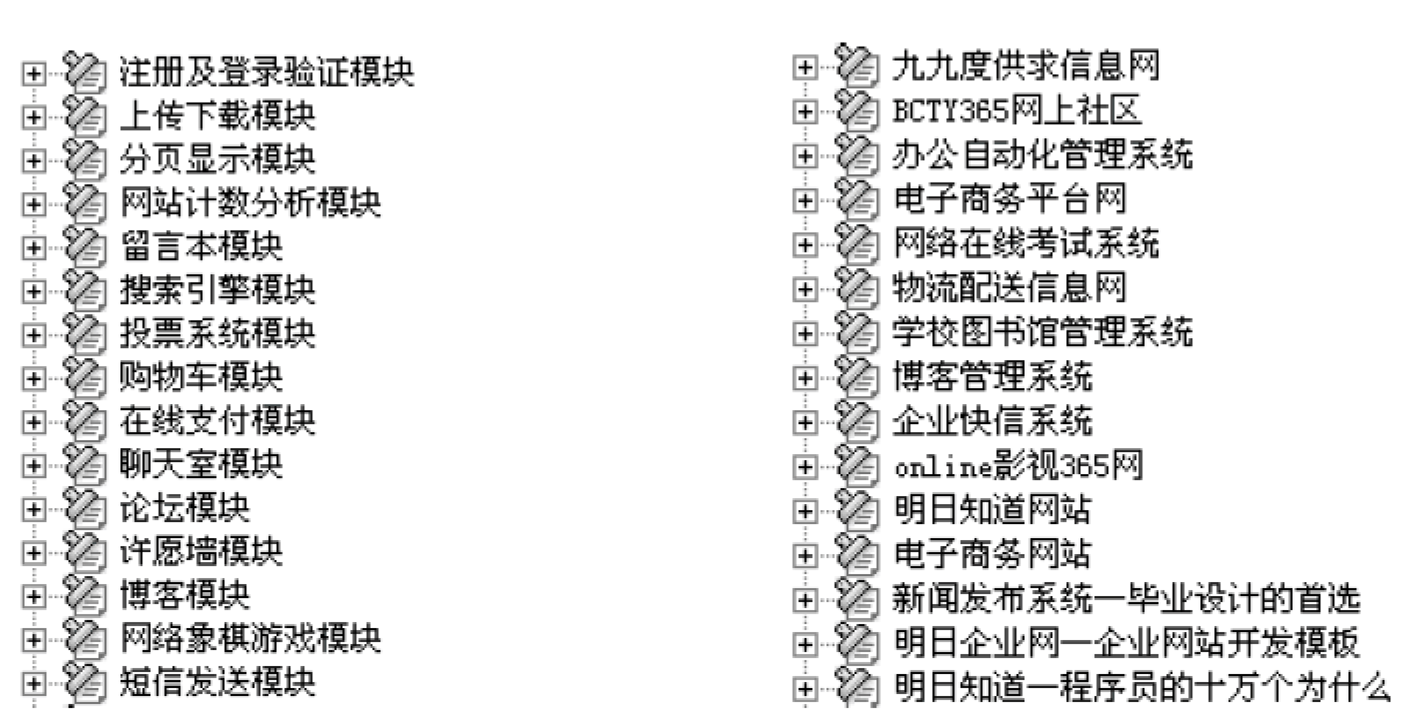
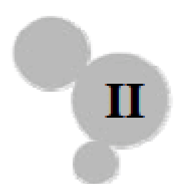


图 5　模块资源库和项目资源库目录

万事俱备，该到软件开发的主战场上接受洗礼了。面试资源库提供了大量国内外软件企业的常见面试真题，同时还提供了程序员职业规划、程序员面试技巧、企业面试真题汇编和虚拟面试系统等精彩内容，是程序员求职面试的绝佳指南。面试资源库的具体内容如图 6 所示。



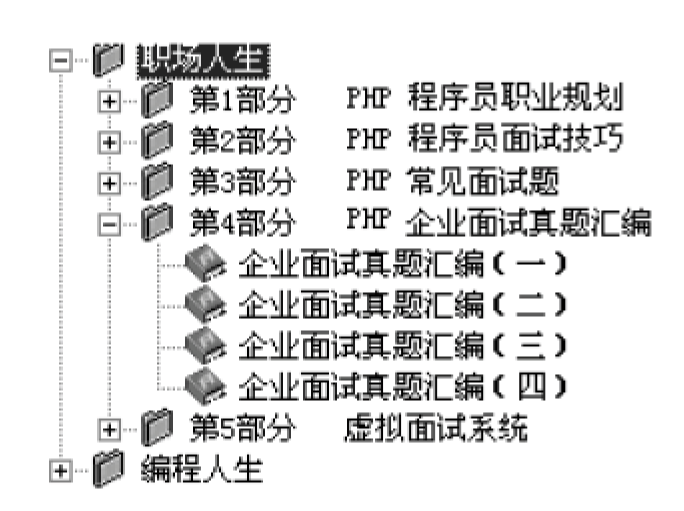


图 6　面试资源库的具体内容

如果您在使用 PHP 开发资源库时遇到问题，可加我们的 QQ：4006751066（可容纳 10 万人），我们将竭诚为您服务。

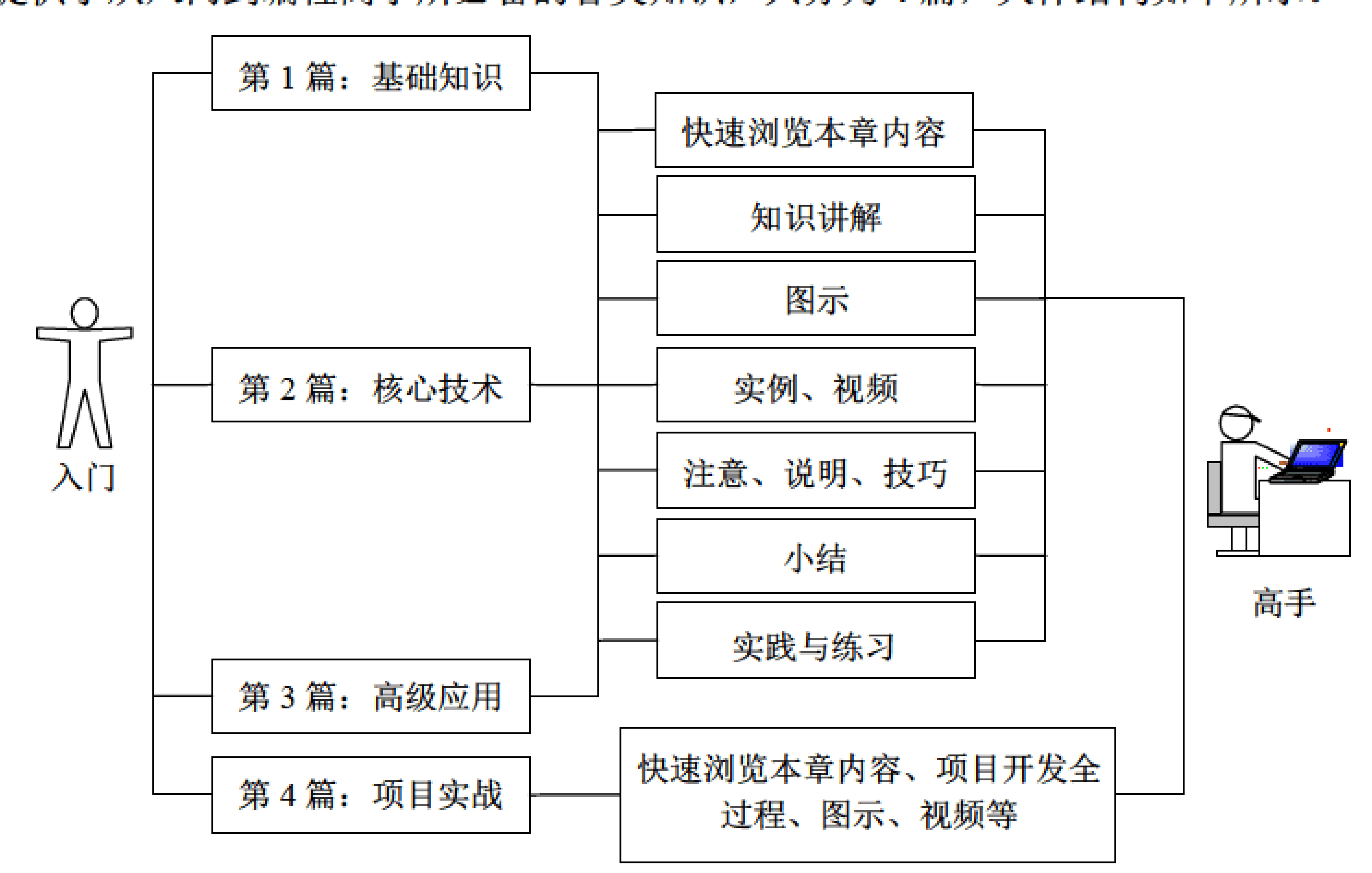
前　言

Preface

MySQL 数据库是世界上最流行的数据库之一。全球最大的网络搜索引擎公司 Google 使用的数据库就是 MySQL，并且国内的很多大型网络公司也选择 MySQL 数据库，如百度、网易和新浪等。据统计，世界上一流的互联网公司中，排名前 20 位的有 80%是 MySQL 的忠实用户。目前，MySQL 已经被列为全国计算机等级考试二级的考试科目。

本书内容

本书提供了从入门到编程高手所必备的各类知识，共分为 4 篇，大体结构如下所示。



第 1 篇：基础知识。本篇通过对数据库基础、初识 MySQL、使用 MySQL 图形化管理工具、数据库操作、存储引擎及数据类型和操作数据表等内容的介绍，并结合大量的图示、举例、视频等使读者快速掌握 MySQL，并为以后的知识奠定坚实的基础。

第 2 篇：核心技术。本篇介绍 MySQL 基础，表数据的增、删、改操作，数据查询，常用函数，索引，视图等内容。学习完这一部分，能够了解和熟悉 MySQL 及常用的函数，使用 SQL 操作 MySQL 数据库中的视图，掌握 SQL 查询、子查询、嵌套查询、连接查询的用法等。

第 3 篇：高级应用。本篇介绍数据完整性约束、存储过程与存储函数、触发器、事务的应用、事件、备份与恢复、MySQL 性能优化、权限管理及安全控制、PHP 管理 MySQL 数据库中的数据等内容。

学习完这一部分，能够掌握如何进行数据的导入与导出操作，以及使用存储过程、触发器、事务、事件等。通过这些内容不仅可以优化查询，还可以提高数据访问速度，更好地维护 MySQL 的权限及其安全。另外，还介绍了应用 PHP 管理 MySQL 数据库中的数据，对于想要使用 PHP 开发的读者非常实用。

第 4 篇：项目实战。本篇分别使用 PHP 和 Java 两种语言，结合 MySQL 实现了两个大型的、完整的管理系统，通过这两个项目，运用软件工程的设计思想，帮助读者学习如何进行软件项目的实践开发。书中按照编写系统分析→系统设计→数据库与数据表设计→公共模块设计→创建项目→实现项目→项目总结的过程进行介绍，带领读者一步一步亲身体验开发项目的全过程。

本书特点

- ☑ **由浅入深，循序渐进：**本书以初、中级程序员为对象，先从 MySQL 基础学起，再学习 MySQL 的核心技术，然后学习 MySQL 的高级应用，最后学习分别使用 PHP 和 Java 等语言结合 MySQL 开发完整项目。讲解过程中步骤详尽，版式新颖，在操作的内容图片上以“❶❷❸…”编号+内容的方式进行标注，让读者在阅读中一目了然，从而快速把握书中内容。
- ☑ **语音视频，讲解详尽：**书中每一章节均提供声图并茂的教学视频，读者可以在光盘中找到相应章节的视频。这些视频能够引导初学者快速入门，感受编程的快乐和成就感，增强进一步学习的信心，从而快速成为编程高手。
- ☑ **实例典型，轻松易学：**通过例子学习是最好的学习方式，本书通过一个知识点、一个例子、一个结果、一段评析和一个综合应用的模式，透彻、详尽地讲述了实际开发中所需的各类知识。
- ☑ **精彩栏目，贴心提醒：**本书根据需要在各章使用了很多“注意”“说明”和“技巧”等小栏目，让读者可以在学习过程中更轻松地理解相关知识点及概念，并轻松地掌握个别技术的应用技巧。
- ☑ **应用实践，随时练习：**书中几乎每章都提供了“实践与练习”，读者能够通过对问题的解答重新回顾、熟悉所学的知识，举一反三，为进一步学习做好充分的准备。

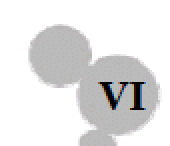
带格式的：项目符号和编号

读者对象

- ☑ 初学编程的自学者
- ☑ 编程爱好者
- ☑ 大中专院校的老师和学生
- ☑ 相关培训机构的老师和学员
- ☑ 毕业设计的学生
- ☑ 初、中级程序开发人员
- ☑ 程序测试及维护人员
- ☑ 参加实习的“菜鸟”程序员

读者服务

为了方便读者，本书提供了学习答疑网站：www.mingribook.com。有关本书的内容均可在网站上留言，作者力求在 24 小时内回复，节假日除外。



致读者

本书由明日科技策划并组织编写，主要编写人员有王国辉、张宝华、王小科、申小琦、董刚、赛奎春、房德山、杨丽、高春艳、辛洪郁、周佳星、张鑫、葛忠月、刘杰、白宏健、张雳霆、马新新、冯春龙、宋万勇、李文欣、王东东、柳琳、王盛鑫、徐明明、杨柳、赵宁、王佳雪、于国良、李磊、李彦骏、王泽奇、贾景波、谭慧、李丹、吕玉翠、孙巧辰、赵颖、江玉贞、周艳梅、房雪坤、裴莹、郭铁、张金辉、王敬杰、高茹、李贺、陈威、高飞、刘志铭、高润岭、于国槐、郭锐、郭鑫、邹淑芳、李根福、杨贵发、王喜平。在编写过程中，我们以科学、严谨的态度，力求精益求精，但疏漏在所难免，敬请广大读者批评指正。我们的服务邮箱是 tmoonbook@sina.com，th_press@263.net。读者在阅读本书时，如果发现或遇到问题，可以发送电子邮件及时与我们联系，我们会尽快给予答复。

感谢您购买本书，希望本书能成为您编程路上的领航者。

“零门槛”编程，一切皆有可能。

祝读书快乐！

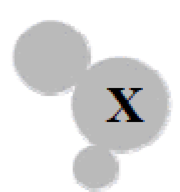
编　者

目　录

Contents

第 1 篇　基 础 知 识

第 2 篇 核 心 技 术



第 3 篇 高 级 应 用

第 4 篇 项 目 实 战



光盘“开发资源库”目录

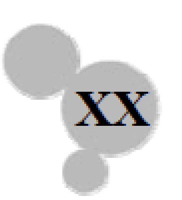
第 1 大部分　实例资源库

（808 个完整实例分析，光盘路径：开发资源库/实例资源库）



第 2 大部分　模块资源库

（15 个经典模块，光盘路径：开发资源库/模块资源库）





第 3 大部分　项目资源库

（15 个企业开发项目，光盘路径：开发资源库/项目资源库）











第 4 大部分　能力测试题库

（626 道能力测试题目，光盘路径：开发资源库/能力测试）

第 5 大部分　面试资源库

（342 项面试真题，光盘路径：开发资源库/面试系统）





基础知识

- 第 1 章　数据库基础
- 第 2 章　初识 MySQL
- 第 3 章　使用 MySQL 图形化管理工具
- 第 4 章　数据库操作
- 第 5 章　存储引擎及数据类型
- 第 6 章　操作数据表

本篇通过对数据库基础、初识 MySQL、使用 MySQL 图形化管理工具、数据库操作、存储引擎及数据类型和操作数据表等内容的介绍，并结合大量的图示、举例和视频等帮助读者快速掌握 MySQL，并为以后的知识奠定坚实的基础。

第 1 章

数据库基础

（视频讲解：25 分钟）

本章主要介绍数据库的相关概念，主要包括数据库系统概述、数据模型和数据库的体系结构。通过本章的学习，读者应该掌握数据库系统、数据模型、数据库三级模式结构以及数据库规范化等概念。

通过阅读本章，读者可以：

- 了解数据库技术的发展史
- 掌握数据库系统的组成
- 熟悉数据模型
- 掌握关系数据库
- 掌握数据库的体系结构

1.1　数据库系统概述

1.1.1　数据库技术的发展

数据库技术是应数据管理任务的需求而产生的，随着计算机技术的发展，对数据管理技术也不断地提出更高的要求，其先后经历了人工管理、文件系统和数据库系统 3 个阶段，下面分别进行介绍。

1. 人工管理阶段

20 世纪 50 年代中期以前，计算机主要用于科学计算。当时硬件和软件设备都很落后，数据基本依赖于人工管理。人工管理阶段具有如下特点。

（1）数据不保存。

（2）使用应用程序管理数据。

（3）数据不共享。

（4）数据不具有独立性。

2. 文件系统阶段

20 世纪 50 年代后期到 20 世纪 60 年代中期，硬件和软件技术都有了进一步发展，有了磁盘等存储设备和专门的数据管理软件（即文件系统）。该阶段具有如下特点。

（1）数据可以长期保存。

（2）由文件系统管理数据。

（3）共享性差，数据冗余大。

（4）数据独立性差。

3. 数据库系统阶段

20 世纪 60 年代后期以来，计算机应用于管理系统，而且规模越来越大，应用越来越广泛，数据量急剧增长，对共享功能的要求越来越强烈，这样使用文件系统管理数据已经不能满足要求，于是出现了数据库系统来统一管理数据。数据库系统的出现，满足了多用户、多应用共享数据的需求，比文件系统具有明显的优点，标志着数据管理技术的飞跃。

1.1.2　数据库系统的组成

数据库系统（DataBase System，DBS）是采用数据库技术的计算机系统，是由数据库（数据）、数据库管理系统、数据库管理员（人员）、支持数据库系统的硬件和软件（应用开发工具、应用系统等）以及用户 5 部分构成的运行实体，如图 1.1 所示。其中，数据库管理员（DataBase Administrator，

DBA）是对数据库进行规划、设计、维护和监视等的专业管理人员，在数据库系统中起着非常重要的作用。

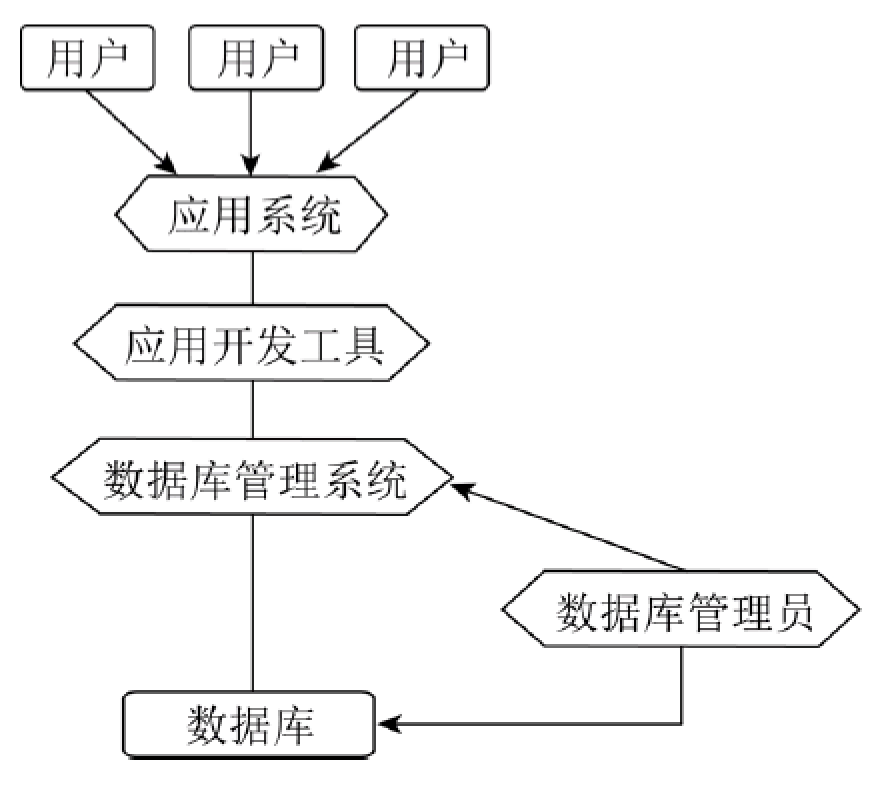


图 1.1　数据库系统的组成

1.2　数 据 模 型

1.2.1　数据模型的概念

数据模型是数据库系统的核心与基础，是关于描述数据与数据之间的联系、数据的语义、数据一致性约束的概念性工具的集合。

数据模型通常是由数据结构、数据操作和完整性约束 3 部分组成的，分别如下。

（1）数据结构：是对系统静态特征的描述，描述对象包括数据的类型、内容、性质和数据之间的相互关系。

（2）数据操作：是对系统动态特征的描述，是对数据库各种对象实例的操作。

（3）完整性约束：是完整性规则的集合，它定义了给定数据模型中数据及其联系所具有的制约和依存规则。

1.2.2　常见的数据模型

常用的数据库数据模型主要有层次模型、网状模型和关系模型，下面分别进行介绍。

（1）层次模型：用树状结构表示实体类型及实体间联系的数据模型称为层次模型，如图 1.2 所示。它具有以下特点。

① 每棵树有且仅有一个无双亲节点，称为根。

② 树中除根外所有节点有且仅有一个双亲。



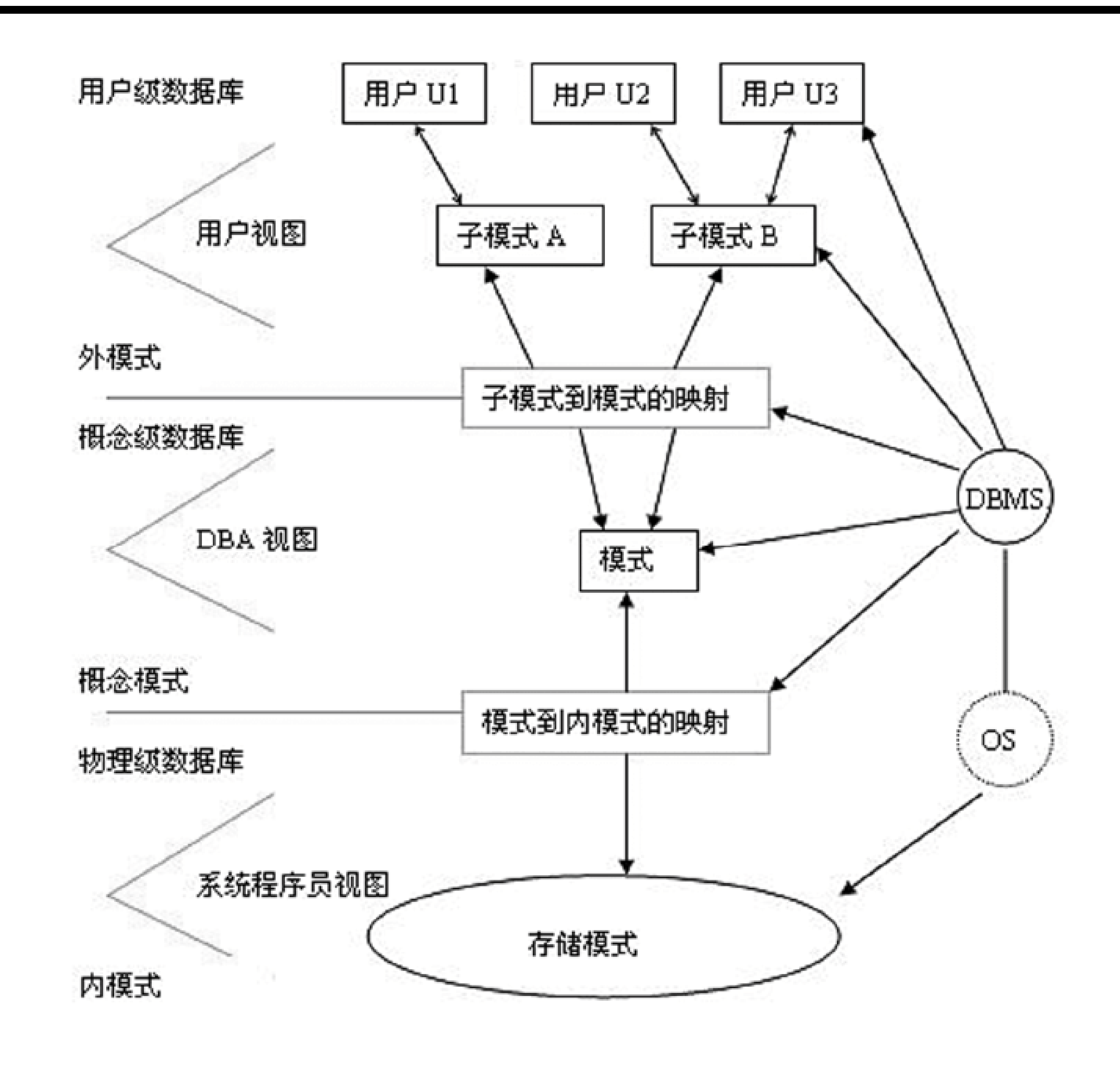


图 1.2　层次模型

（2）网状模型：用有向图结构表示实体类型及实体间联系的数据模型称为网状模型，如图 1.3 所示。用网状模型编写应用程序极其复杂，数据的独立性较差。

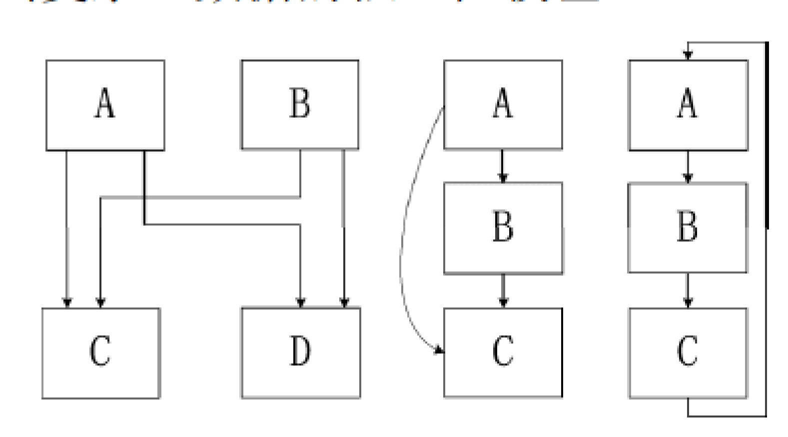


图 1.3　网状模型

（3）关系模型：以二维表来描述数据。关系模型中，每个表有多个字段列和记录行，每个字段列有固定的属性（数字、字符、日期等），如图 1.4 所示。关系模型数据结构简单、清晰、具有很高的数据独立性，是目前主流的数据库数据模型。

关系模型的基本术语如下。

① 关系：一个二维表就是一个关系。

② 元组：二维表中的一行，即表中的记录。

③ 属性：二维表中的一列，用类型和值表示。

④ 域：每个属性取值的变化范围，如性别的域为{男，女}。

关系中的数据约束如下。

① 实体完整性约束：约束关系的主键中属性值不能为空值。

② 参照完整性约束：关系之间的基本约束。

③ 用户定义的完整性约束：反映了具体应用中数据的语义要求。

学生信息表

学生姓名	年级	家庭住址
张三	2000	成都
李四	2000	北京
王五	2000	上海

成绩表

学生姓名	课程	成绩
张三	数学	100
张三	物理	95
张三	社会	90
李四	数学	85
李四	社会	90
王五	数学	80
王五	物理	75

图 1.4　关系模型

1.2.3　关系数据库的规范化

关系数据库的规范化理论为：关系数据库中的每一个关系都要满足一定的规范。根据满足规范的条件不同，可以分为 5 个等级：第一范式（1NF）、第二范式（2NF）……第五范式（5NF）。其中，NF 是 Normal Form 的缩写。一般情况下，只要把数据规范到第三范式标准就可以满足需要了。下面举例介绍前 3 种范式。

1. 第一范式（1NF）

在一个关系中，消除重复字段，且各字段都是最小的逻辑存储单位。第一范式是第二和第三范式的基础，是最基本的范式。第一范式包括下列指导原则。

（1）数据组的每个属性只可以包含一个值。

（2）关系中的每个数组必须包含相同数量的值。

（3）关系中的每个数组一定不能相同。

在任何一个关系数据库中，第一范式是对关系模式的基本要求，不满足第一范式的数据库就不是关系型数据库。

如果数据表中的每一个列都是不可再分割的基本数据项——即同一列中不能有多个值，那么就称此数据表符合第一范式，由此可见第一范式具有不可再分解的原子特性。

在第一范式中，数据表的每一个行只包含一个实体的信息，并且每一行的每一列只能存放实体的一个属性。例如，对于学生信息，不可以将学生实体的所有属性信息（如学号、姓名、性别、年龄、



班级等）都放在一个列中显示，也不能将学生实体的两个或多个属性信息放在一个列中显示，学生实体的每个属性信息都放在一个列中显示。

如果数据表中的列信息都符合第一范式，那么在数据表中的字段都是单一的、不可再分的。如表 1.1 就是不符合第一范式的学生信息表，因为“班级”列中包含“系别”和“班级”两个属性信息，这样“班级”列中的信息就不是单一的，是可以再分的；而表 1.2 就是符合第一范式的学生信息表，它将原“班级”列的信息拆分到“系别”列和“班级”列中。

表 1.1　不符合第一范式的学生信息表

学　号	姓　名	性　别	年　龄	班　级
9527	东*方	男	20	计算机系 3 班

表 1.2　符合第一范式的学生信息表

学　号	姓　名	性　别	年　龄	系　别	班　级
9527	东*方	男	20	计算机	3 班

2. 第二范式（2NF）

第二范式是在第一范式的基础上建立起来的，即满足第二范式必先满足第一范式（1NF）。第二范式要求数据库表中的每个实体（即各个记录行）必须可以被唯一地区分。为实现区分各行记录通常需要为表设置一个“区分列”，用以存储各个实体的唯一标识。在学生信息表中，设置了“学号”列，由于每个学生的编号都是唯一的，因此每个学生可以被唯一地区分（即使学生存在重名的情况下），那么这个唯一属性列被称为主关键字或主键。

第二范式要求实体的属性完全依赖于主关键字，即不能存在仅依赖主关键字一部分的属性，如果存在，那么这个属性和主关键字的这一部分应该分离出来形成一个新的实体，新实体与原实体之间是一对多的关系。

例如，这里以“员工工资信息表”为例，若以（员工编码、岗位）为组合关键字（即复合主键），就会存在如下决定关系。

```
（员工编码，岗位）→（决定）（姓名、年龄、学历、基本工资、绩效工资、奖金）
```

在上面的决定关系中，还可以进一步拆分为如下两种决定关系。

```
（员工编码）→（决定）（姓名、年龄、学历）
（岗位）→（决定）（基本工资）
```

其中，员工编码决定了员工的基本信息（包括姓名、年龄、学历等）；而岗位决定了基本工资，所以这个关系表不满足第二范式。

对于上面的这种关系，可以把上述两个关系表更改为如下 3 个表。

员工档案表：EMPLOYEE（员工编码、姓名、年龄和学历）

岗位工资表：QUARTERS（岗位和基本工资）

员工工资表：PAY（员工编码、岗位、绩效工资和奖金）

3. 第三范式（3NF）

第三范式是在第二范式的基础上建立起来的，即满足第三范式必先满足第二范式。第三范式要求关系表不存在非关键字列对任意候选关键字列的传递函数依赖，也就是说，第三范式要求一个关系表中不包含已在其他表中包含的非主关键字信息。

所谓传递函数依赖，就是指如果存在关键字段 A 决定非关键字段 B，而非关键字段 B 决定非关键字段 C，则称非关键字段 C 传递函数依赖于关键字段 A。

例如，这里以员工信息表（EMPLOYEE）为例，该表中包含员工编号、员工姓名、年龄、部门编码、部门经理等信息，该关系表的关键字为“员工编号”，因此存在如下决定关系。

```
（员工编码）→（决定）（员工姓名、年龄、部门编码、部门经理）
```

上面的这个关系表是符合第二范式的，但它不符合第三范式，因为该关系表内部隐含着如下决定关系。

```
（员工编码）→（决定）（部门编码）→（决定）（部门经理）
```

上面的关系表存在非关键字段“部门经理”对关键字段“员工编码”的传递函数依赖。对于上面的这种关系，可以把这个关系表（EMPLOYEE）更改为如下两个关系表。

员工信息表：EMPLOYEE（员工编码、员工姓名、年龄和部门编码）

部门信息表：DEPARTMENT（部门编码和部门经理）

对于关系型数据库的设计，理想的设计目标是按照“规范化”原则存储数据，因为这样做能够消除数据冗余、更新异常、插入异常和删除异常。

1.2.4　关系数据库的设计原则

数据库设计是指对于一个给定的应用环境，根据用户的需求，利用数据模型和应用程序模拟现实世界中该应用环境的数据结构和处理活动的过程。

数据库设计原则如下。

（1）数据库内数据文件的数据组织应获得最大限度的共享、最小的冗余度，消除数据及数据依赖关系中的冗余部分，使依赖于同一个数据模型的数据达到有效的分离。

（2）保证输入、修改数据时数据的一致性与正确性。

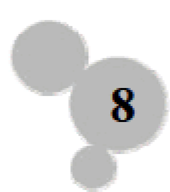
（3）保证数据与使用数据的应用程序之间的高度独立性。

1.2.5　实体与关系

实体是指客观存在并可相互区别的事物，实体既可以是实际的事物，也可以是抽象的概念或关系。

实体之间有 3 种关系，分别如下。

（1）一对一关系：是指表 A 中的一条记录在表 B 中有且只有一条相匹配的记录。在一对一关系中，大部分相关信息都在一个表中。



（2）一对多关系：是指表 A 中的行可以在表 B 中有许多匹配行，但是表 B 中的行只能在表 A 中有一个匹配行。

（3）多对多关系：是指关系中每个表的行在相关表中具有多个匹配行。在数据库中，多对多关系的建立是依靠第三个表（称作连接表）实现的，连接表包含相关的两个表的主键列，然后从两个相关表的主键列分别创建与连接表中的匹配列的关系。

1.3 数据库的体系结构

1.3.1 数据库三级模式结构

数据库的三级模式结构是指模式、外模式和内模式。

1. 模式

模式也称逻辑模式或概念模式，是数据库中全体数据的逻辑结构和特征的描述，是所有用户的公共数据视图。一个数据库只有一个模式。模式处于三级结构的中间层。

注意

定义模式时不仅要定义数据的逻辑结构，而且要定义数据之间的联系，定义与数据有关的安全性、完整性要求。

2. 外模式

外模式也称用户模式，是数据库用户（包括应用程序员和最终用户）能够看见和使用的局部数据的逻辑结构和特征的描述，是数据库用户的数据视图，是与某一应用有关的数据的逻辑表示。外模式是模式的子集，一个数据库可以有多个外模式。

说明

外模式是保证数据安全性的一个有力措施。

3. 内模式

内模式也称存储模式，是数据物理结构和存储方式的描述，是数据在数据库内部的表示方式。一个数据库只有一个内模式。

1.3.2 三级模式之间的映射

为了能够在内部实现数据库的 3 个抽象层次的联系和转换，数据库管理系统在三级模式之间提供了两层映射，分别为外模式/模式映射和模式/内模式映射，下面分别介绍。

1．外模式/模式映射

对于同一个模式可以有任意多个外模式。对于每一个外模式，数据库系统都有一个外模式/模式映射。当模式改变时，由数据库管理员对各个外模式/模式映射做相应的改变，可以使外模式保持不变。这样，依据数据外模式编写的应用程序就不用修改，保证了数据与程序的逻辑独立性。

2．模式/内模式映射

数据库中只有一个模式和一个内模式，所以模式/内模式映射是唯一的，它定义了数据库的全局逻辑结构与存储结构之间的对应关系。当数据库的存储结构改变时，由数据库管理员对模式/内模式映射做相应改变，可以使模式保持不变，应用程序相应地也不做变动。这样，保证了数据与程序的物理独立性。

1.4 小　　结

本章主要介绍的是数据库技术中的一些基本概念和原理，其中重点包括数据库技术的发展、数据库系统的组成、数据模型的概念、常见的数据模型、关系数据库的规范化及设计原则、实体与关系、数据库的三级模式结构，以及三级模式之间的映射等内容。其中，常见的数据模型和关系数据库的规范化及设计原则希望读者认真学习，重点掌握。

1.5 实践与练习

1．数据库技术的发展经历了哪 3 个阶段？
2．数据模型通常是由哪 3 部分组成的？
3．常用的数据库数据模型主要有哪几种？

第 2 章

初识 MySQL

（视频讲解：12 分钟）

MySQL 数据库可以称得上是目前运行速度最快的 SQL 数据库。除了具有许多其他数据库所不具备的功能和选择之外，MySQL 数据库还是一种完全免费的产品，用户可以直接从网上下载使用，不必支付任何费用。另外，MySQL 数据库的跨平台性也是一大优势。本章将对 MySQL 数据库的概念、MySQL 的特性、应用环境，以及如何安装、配置、启动、连接、断开和停止 MySQL 服务器进行详细介绍。

通过阅读本章，读者可以：

- 了解 MySQL 数据库的概念及其优势
- 熟悉 MySQL 的特性
- 了解 MySQL 的应用环境
- 掌握如何安装和配置 MySQL 服务器
- 掌握启动、连接、断开和停止 MySQL 服务器的方法
- 了解如何学好 MySQL

2.1　了解 MySQL

MySQL 是目前最为流行的开放源代码的数据库管理系统，是完全网络化的、跨平台的关系型数据库系统，它是由瑞典的 MySQL AB 公司开发的，由 MySQL 的初始开发人员 David Axmark 和 Michael "Monty" Widenius 于 1995 年建立，目前属于 Oracle 公司。它的象征符号是一只名为 Sakila 的海豚，代表着 MySQL 数据库和团队的速度、能力、精确和优秀本质。

MySQL 数据库可以称得上是目前运行速度最快的 SQL 数据库。除了具有许多其他数据库所不具备的功能和选择之外，MySQL 数据库还是一种完全免费的产品，用户可以直接从网上下载使用，不必支付任何费用。

2.1.1　MySQL 数据库的概念

数据库（Database）就是一个存储数据的仓库。为了方便数据的存储和管理，它将数据按照特定的规律存储在磁盘上。通过数据库管理系统，可以有效地组织和管理存储在数据库中的数据。MySQL 就是这样的一个关系型数据库管理系统（RDBMS），它可以称得上是目前运行速度最快的 SQL 数据库管理系统。

2.1.2　MySQL 的优势

MySQL 是一款自由软件，任何人都可以从其官方网站下载。MySQL 是一个真正的多用户、多线程 SQL 数据库服务器。它是以客户/服务器结构的实现，由一个服务器守护程序 mysqld 和很多不同的客户程序和库的组成。它能够快捷、有效和安全地处理大量的数据。相对于 Oracle 等数据库来说，MySQL 在使用时非常简单。MySQL 的主要目标是快捷、便捷和易用。

MySQL 被广泛地应用在 Internet 上的中小型网站中。由于其体积小、速度快、总体拥有成本低，尤其是开放源代码这一特点，成为多数中小型网站为了降低网站总体拥有成本而选择 MySQL 作为网站数据库的重要指标。

2.1.3　MySQL 的发展史

MySQL 这个名字的由来已经无从考究了。基本指南以及大量的库和工具采用前缀 My，已经有 10 年以上了；另外，MySQL 的创始人之一的 Monty Widenius 的女儿也叫 My。到底哪个是 MySQL 名字的由来，至今仍是一个谜，包括开发者也不知道。

MySQL 的海豚徽标的名字为 Sakila，它是由 MySQL AB 公司的创办人从用户在“Dolphin 命名”比赛中提供的众多建议中选定的，是由来自非洲斯威士兰的开放源码软件开发人 Ambrose Twebaze 提

出的。根据 Ambrose 的说法，按斯威士兰的本地语言，女性化名称 Sakila 源自 SiSwati。Sakila 也是坦桑尼亚、Arusha 地区的一个镇的镇名，靠近 Ambrose 的母国乌干达。

MySQL 从无到有，到技术的不断更新、版本的不断升级，经历了一个漫长的过程，这个过程是实践的过程，是 MySQL 成长的过程。时至今日，MySQL 的版本已经更新到了 MySQL 5.6。如图 2.1 所示为 MySQL 官方网站上的截图，足以反映出 MySQL 的成长历程。

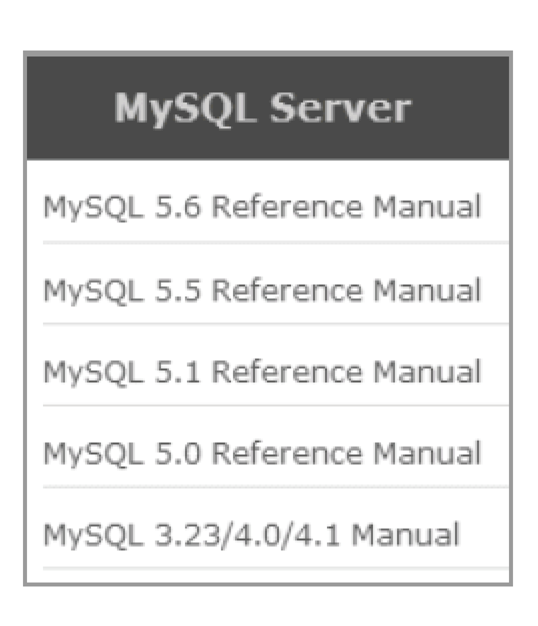


图 2.1　MySQL 版本的发展



2008 年 1 月 16 日，MySQL 被 Sun 公司收购，而到 2009 年，Sun 公司被世界第二大软件供应商 Oracle 公司收购，成为 Oracle 数据库的有益补充。

2.2　MySQL 的特性

MySQL 是一个真正的多用户、多线程 SQL 数据库服务器。SQL（结构化查询语言）是世界上最流行的和标准化的数据库语言。下面看一下 MySQL 的特性。

（1）使用 C 和 C++语言编写，并使用了多种编译器进行测试，保证源代码的可移植性。

（2）支持 AIX、FreeBSD、HP-UX、Linux、Mac OS、Novell Netware、OpenBSD、OS/2 Wrap、Solaris、Windows 等多种操作系统。

（3）为多种编程语言提供了 API。这些编程语言包括 C、C++、Python、Java、Perl、PHP、Eiffel、Ruby 和 Tcl 等。

（4）支持多线程，充分利用 CPU 资源。

（5）优化的 SQL 查询算法，有效地提高查询速度。

（6）既能够作为一个单独的应用程序应用在客户端服务器网络环境中，也能够作为一个库而嵌入到其他软件中提供多语言支持，常见的编码如中文的 GB2312、BIG5，日文的 Shift_JIS 等都可以用作数据表名和数据列名。

（7）提供 TCP/IP、ODBC 和 JDBC 等多种数据库连接途径。

（8）提供用于管理、检查、优化数据库操作的管理工具。

（9）可以处理拥有上千万条记录的大型数据库。

目前的最新版本是 MySQL 5.6，它提供了一组专用功能集，在当今现代化、多功能处理硬件和软件以及中间件构架涌现的环境中，极大地提高了 MySQL 的性能、可扩展性、可用性。

MySQL 5.6 融合了 MySQL 数据库和 InnoDB 存储引擎的优点，能够提供高性能的数据管理解决方案，包括以下几点。

（1）InnoDB 作为默认的数据库存储引擎。

（2）提升了 Windows 系统下的系统性能和可扩展性。

（3）改善性能和可扩展性，全面利用各平台现代多核构架的计算能力。

（4）提高实用性。

（5）提高易管理性和效率。

（6）提高可用性。

（7）改善检测与诊断性能。

2.3 MySQL 的应用环境

MySQL 与其他大型数据库（如 Oracle、DB2、SQL Server 等）相比，确有不足之处，如规模小、功能有限等，但是这丝毫也没有减少它受欢迎的程度。对于个人使用者和中小型企业来说，MySQL 提供的功能已经绰绰有余，而且由于 MySQL 是开放源代码软件，因此可以大大降低总体拥有成本。

目前 Internet 上流行的网站构架方式是 LAMP（Linux+Apache+MySQL+PHP），即使用 Linux 作为操作系统，Apache 作为 Web 服务器，MySQL 作为数据库，PHP 作为服务器端脚本解释器。由于这 4 个软件都是免费或开放源代码软件（FLOSS），因此使用这种方式不用花一分钱（除人工成本）就可以建立起一个稳定、免费的网站系统。

2.4 MySQL 服务器的安装和配置

MySQL 是目前最为流行的开放源码的数据库，是完全网络化的跨平台的关系型数据库系统，它是由 MySQL AB 公司开发、发布并支持的。任何人都能从 Internet 上下载 MySQL 软件，而无须支付任何费用，并且“开放源代码”意味着任何人都可以使用和修改该软件，如果愿意，用户也可以研究源代码并进行恰当的修改，以满足自己的需求，不过需要注意的是，这种“自由”是有范围的。

2.4.1 MySQL 服务器下载

MySQL 服务器的安装包可以到 Oracle 官网（http://www.oracle.com/index.html）中下载。下载 MySQL 的具体步骤如下。



（1）在浏览器的地址栏中输入 URL 地址 http://www.oracle.com/index.html，进入 Oracle 官网首页，将鼠标移动到 Downloads 上，将显示如图 2.2 所示的 Downloads 子菜单。

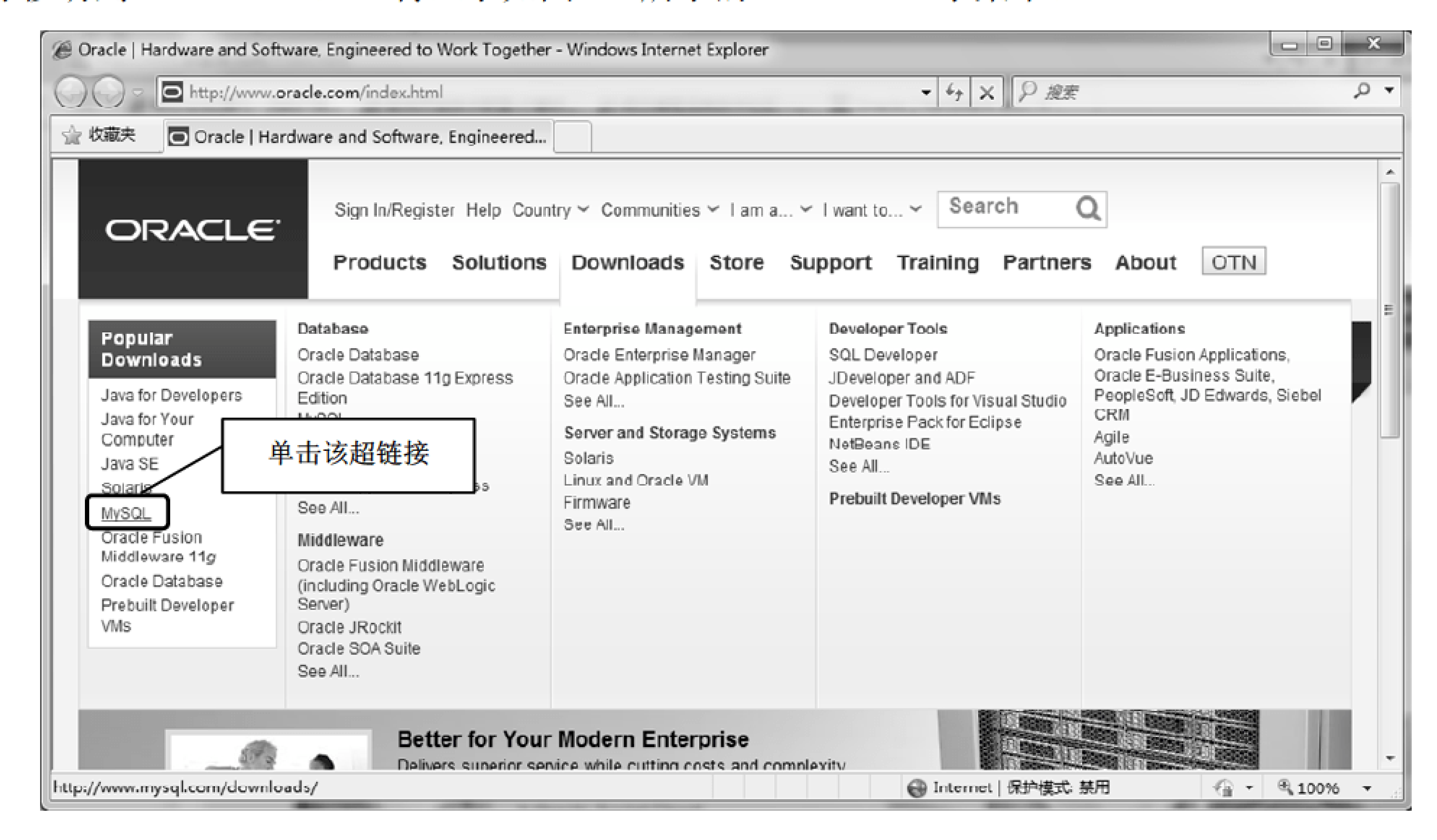


图 2.2　Oracle 官网的 Downloads 子菜单

（2）在如图 2.2 所示的菜单中，单击 MySQL 超链接，进入到 MySQL Downloads 页面，将页面滚动到底部，如图 2.3 所示。



图 2.3　MySQL Downloads 页面

（3）单击 Community (GPL) Downloads >>超链接，进入到 MySQL Community Downloads 页面，如图 2.4 所示。

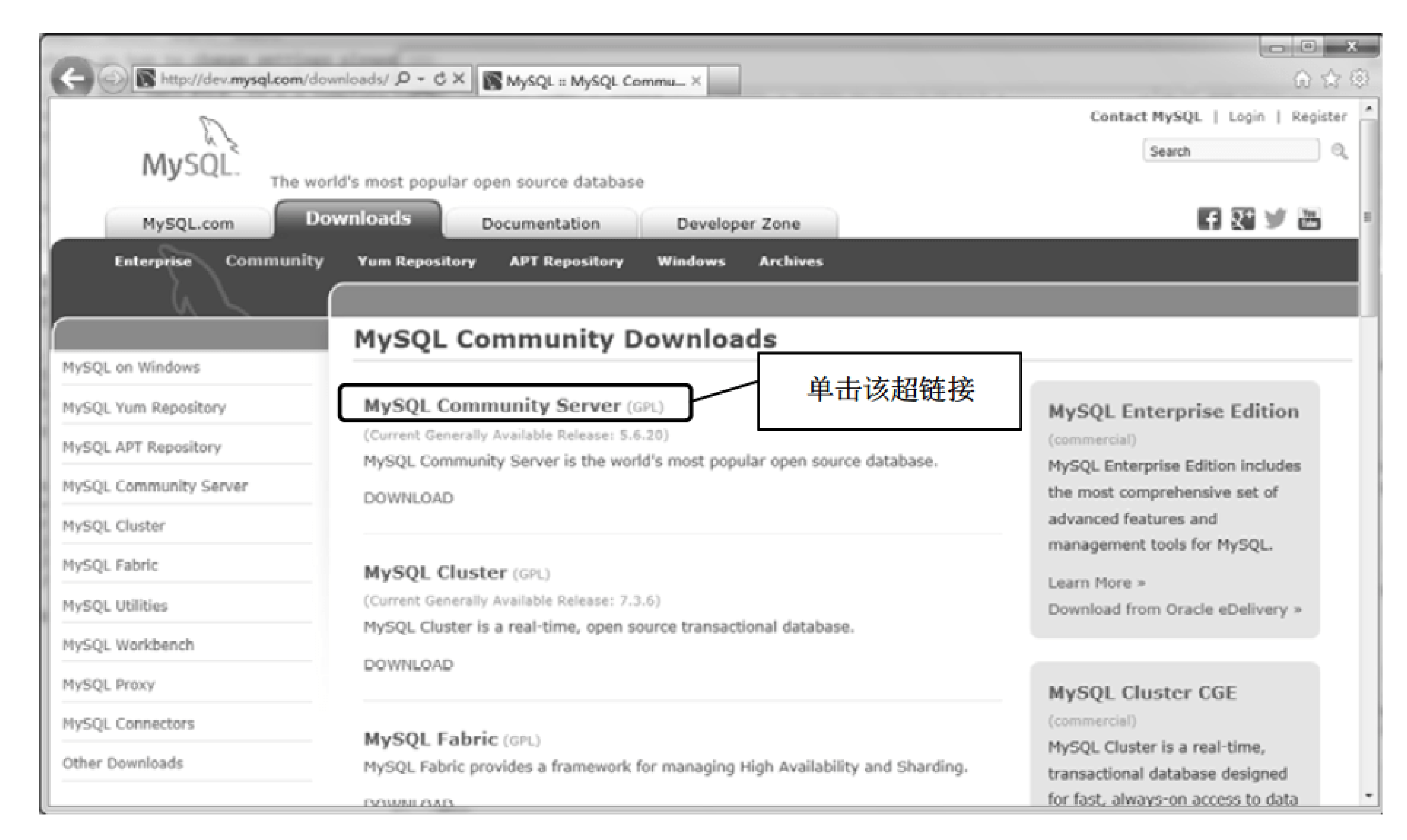


图 2.4 MySQL Community Downloads 页面

（4）单击 MySQL Community Server(GPL)超链接，将进入到 Download MySQL Community Server 页面，将页面滚动到如图 2.5 所示的位置。



图 2.5 Download MySQL Community Server 页面

（5）根据自己的操作系统来选择合适的安装文件，这里以针对 Windows 32 位操作系统的完整版 MySQL Server 为例进行介绍，单击图 2.5 中的图片，将进入到 Download MySQL Installer 页面，在该



页面中，滚动到如图 2.6 所示的位置。



图 2.6　Download MySQL Installer 页面

（6）单击 Download 按钮，将进入到如图 2.7 所示的 Begin Your Download - mysql-installer-community-5.6.20.0.msi 页面。

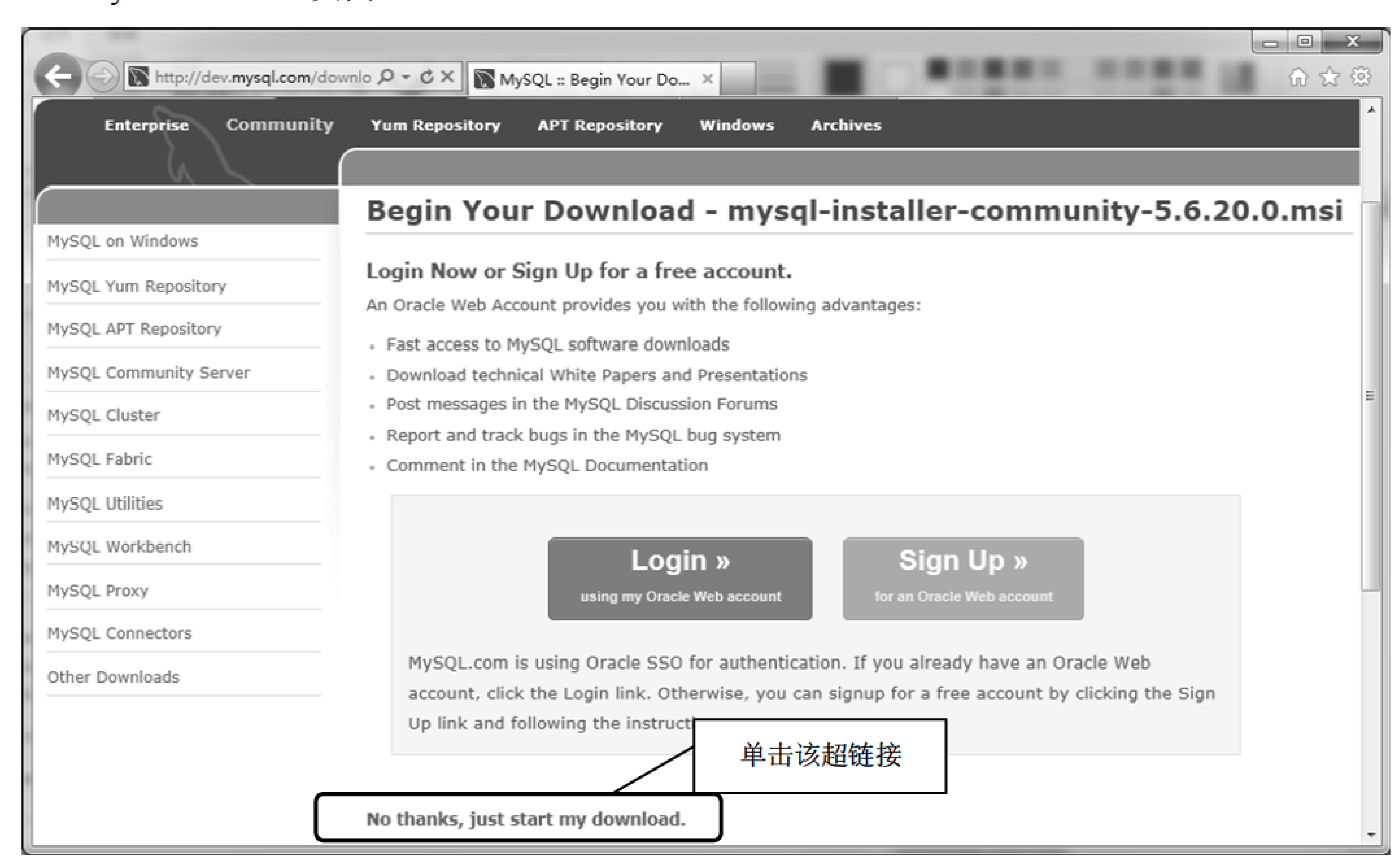


图 2.7　Begin Your Download - mysql-installer-community-5.6.20.0.msi 页面

（7）单击 No thanks, just start my download.超链接，将打开如图 2.8 所示的文件下载对话框，单击“保存”按钮，下载安装文件。

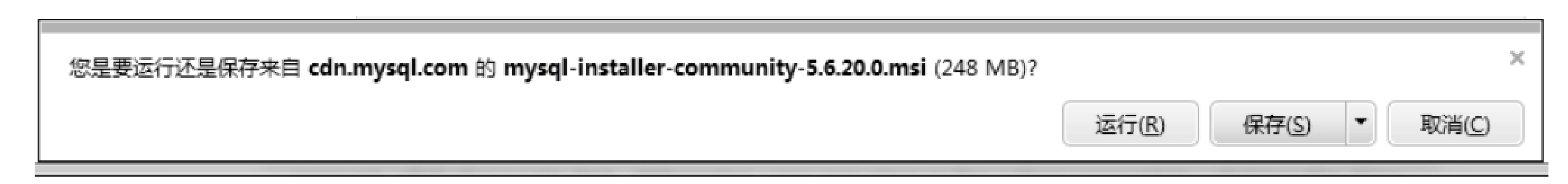


图 2.8　文件下载

2.4.2　MySQL 服务器安装

下载 MySQL 服务器的安装文件以后，将得到一个名称为 mysql-installer-community-5.6.20.0.msi 的安装文件，双击该文件可以进行 MySQL 服务器的安装，具体的安装步骤如下。

（1）双击下载后的 mysql-installer-community-5.6.20.0.msi 文件，打开安装向导，如果没有打开安装向导，而是弹出如图 2.9 所示的对话框，那么还需要先安装.NET 4.0 框架，然后再重新安装双击下载后的安装文件，打开安装向导对话框，如图 2.10 所示。

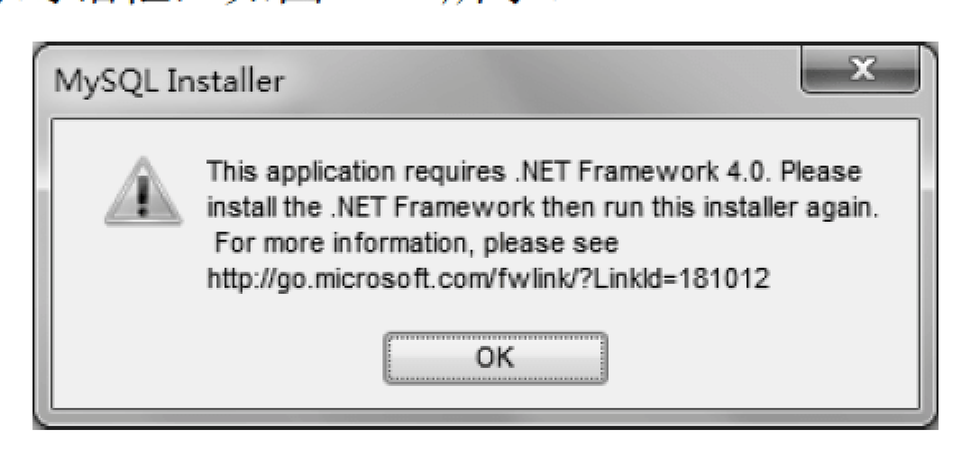


图 2.9　打开需要安装.NET 4.0 框架的提示对话框

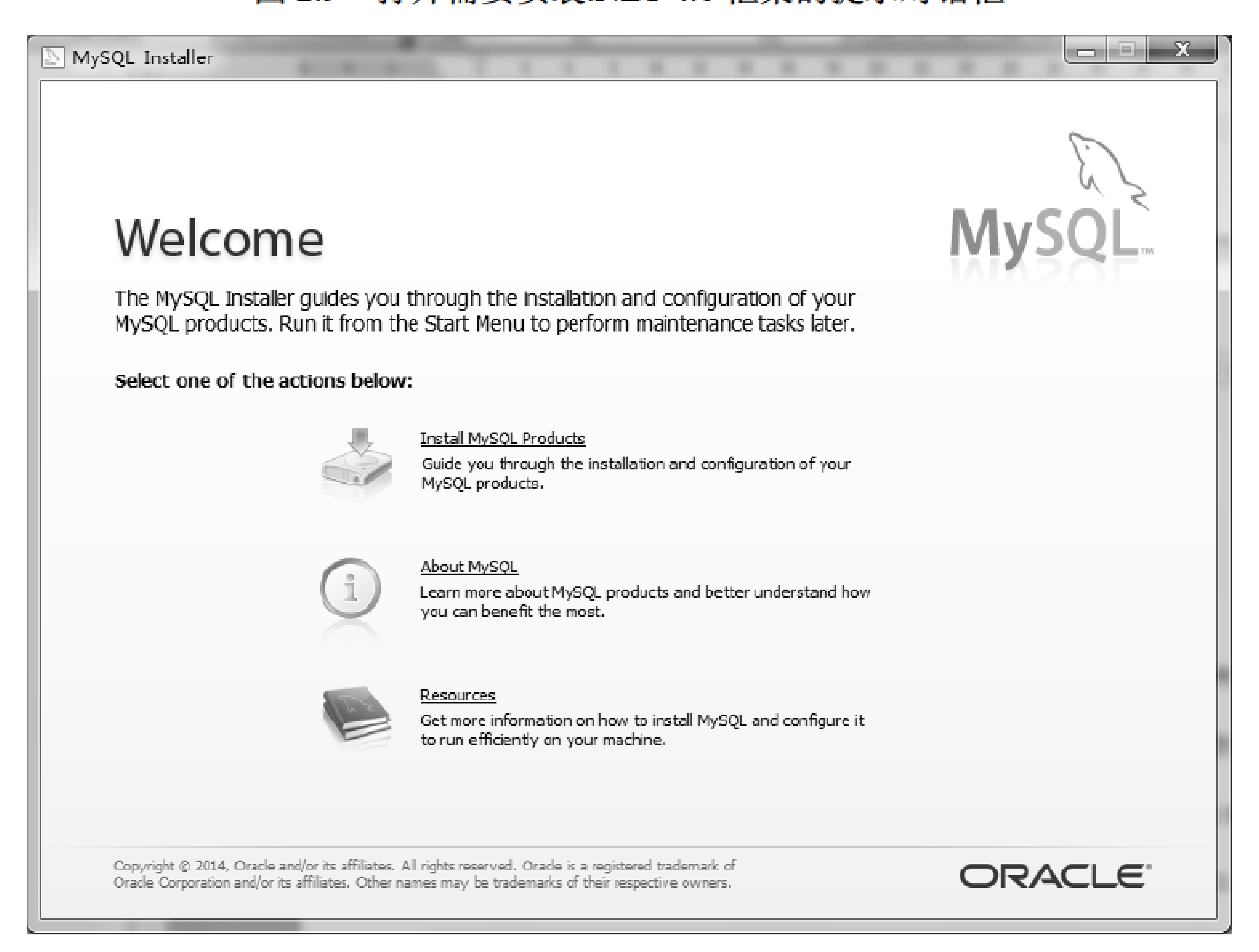


图 2.10　安装向导对话框

（2）在打开的安装向导对话框中单击 Install MySQL Products 超链接，将打开 License Agreement 对话框，询问是否接受协议，选中 I accept the license terms 复选框，接受协议，如图 2.11 所示。



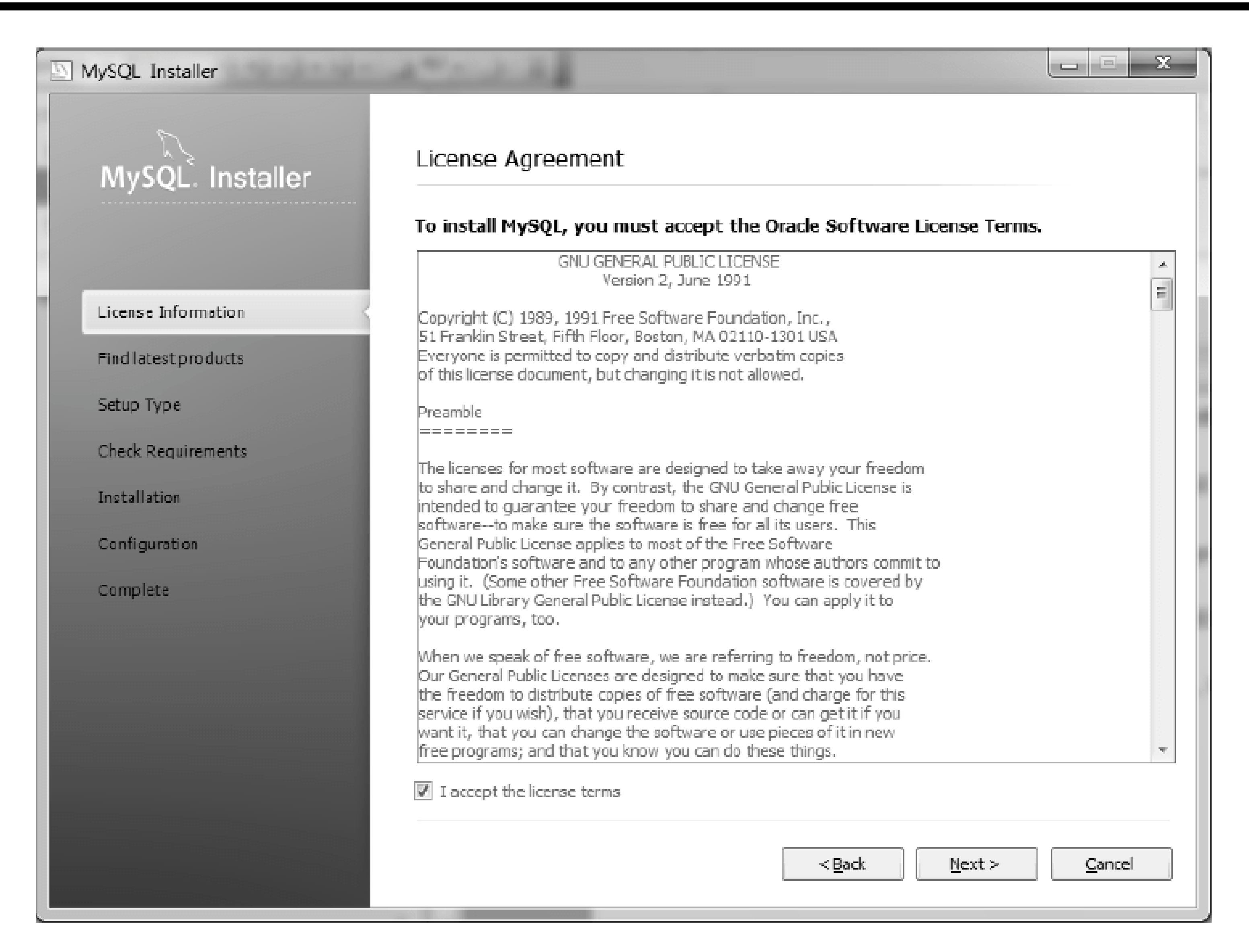


图 2.11　License Agreement 对话框

（3）单击 Next 按钮，将打开 Find latest products 对话框。在该对话框中，选中 Skip the check for updates(not recommended)复选框，这时，原来的 Execute 按钮将转换为 Next 按钮，如图 2.12 所示。

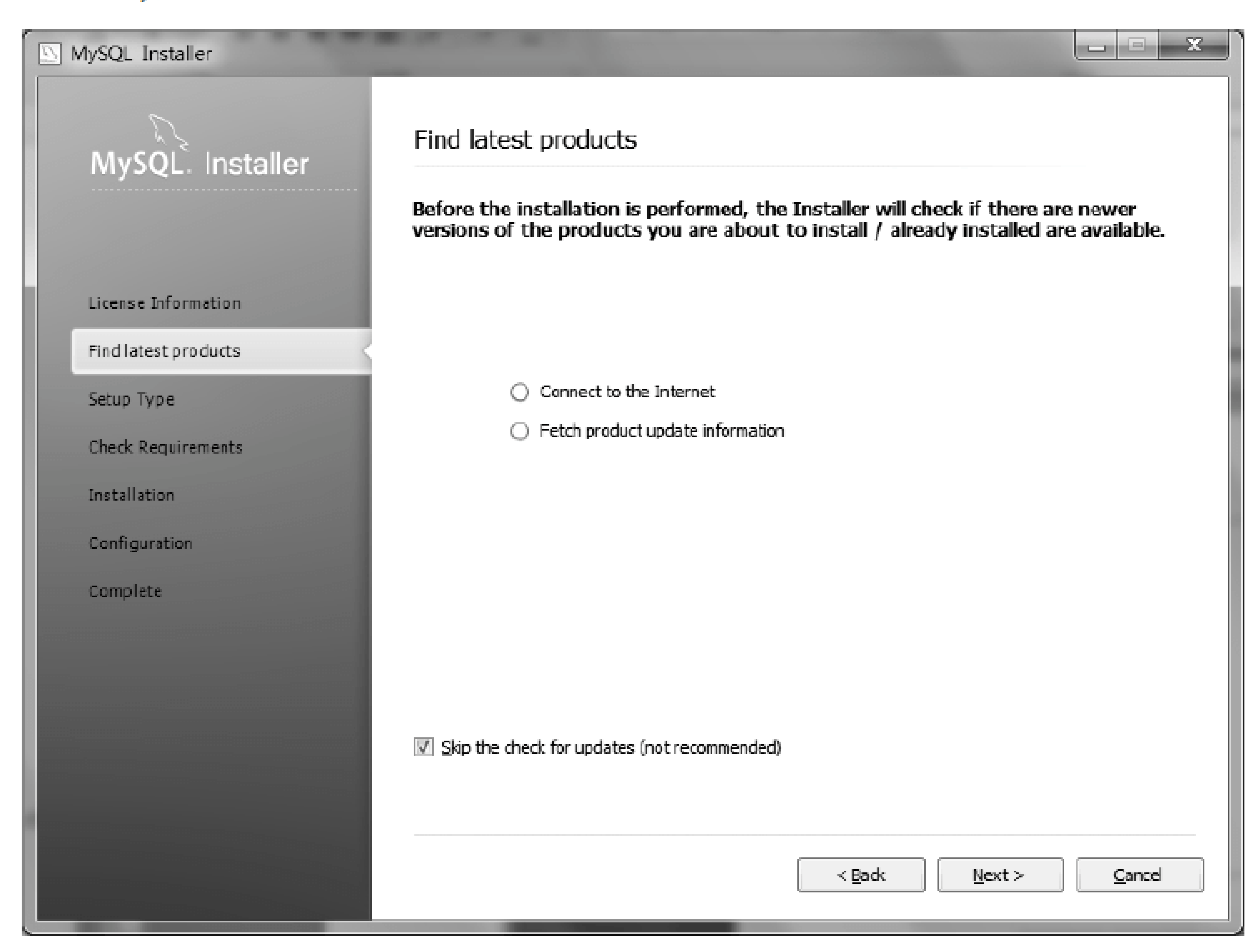


图 2.12　Find latest products 对话框

（4）单击 Next 按钮，将打开 Choosing a Setup Type 对话框，在该对话框中，共包括 Developer Default（开发者默认）、Server only（仅服务器）、Client only（仅客户端）、Full（完全）和 Custom（自定义）5 种安装类型，这里选择 Developer Default，并且将安装路径修改为“C:\Program Files\MySQL\”，数据存放路径修改为“C:\ProgramData\MySQL\MySQL Server 5.6\”，如图 2.13 所示。

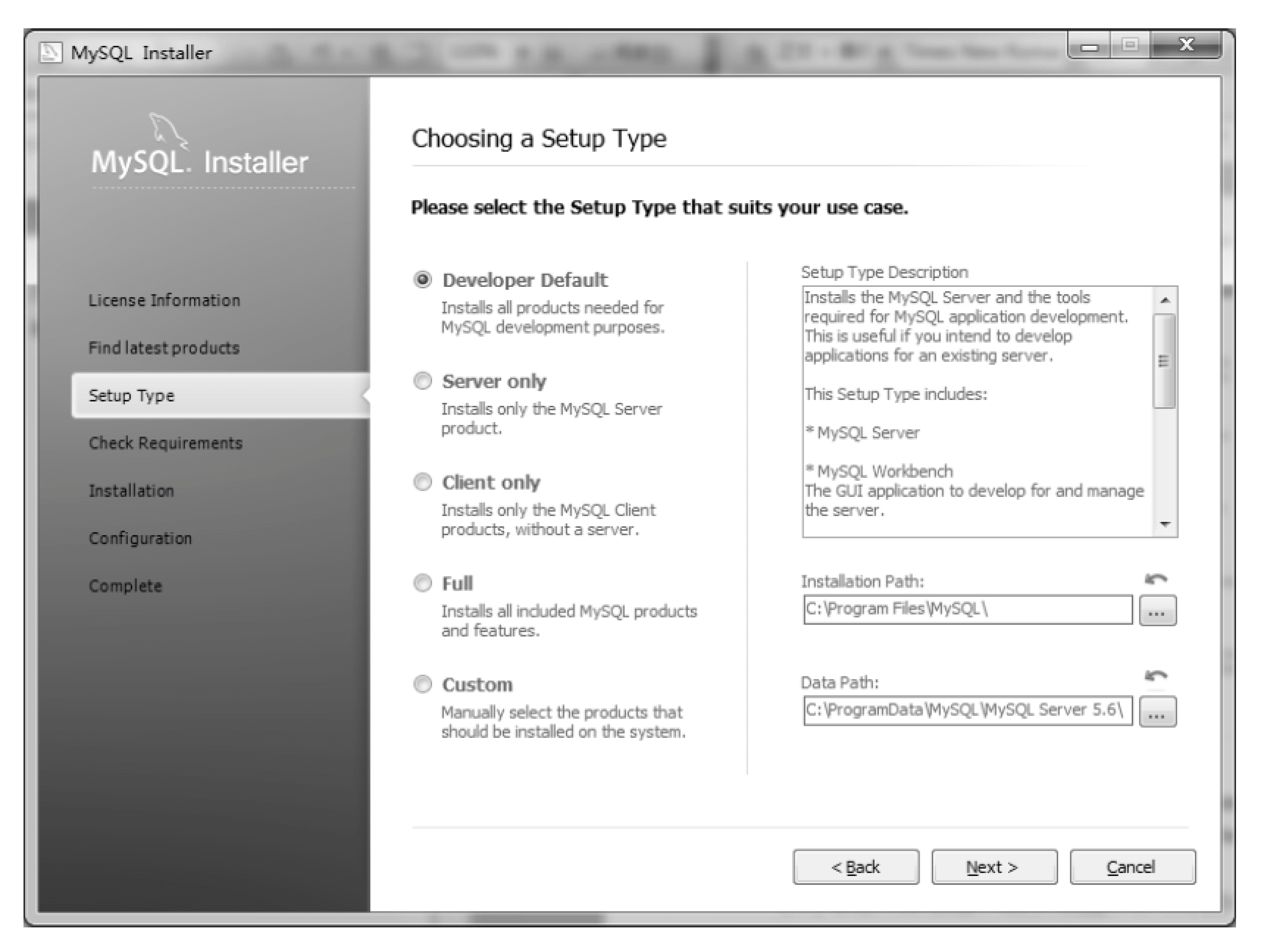


图 2.13　Choosing a Setup Type 对话框

（5）单击 Next 按钮，将打开如图 2.14 所示的 Check Requirements 对话框，在该对话框中检查系统是否具备安装所必需的.NET 4.0 框架和 Microsoft Visual C++ 2010 32-bit runtime，如果不存在，单击 Execute 按钮，将在线安装所需插件，安装完成后，将显示如图 2.15 所示的对话框。

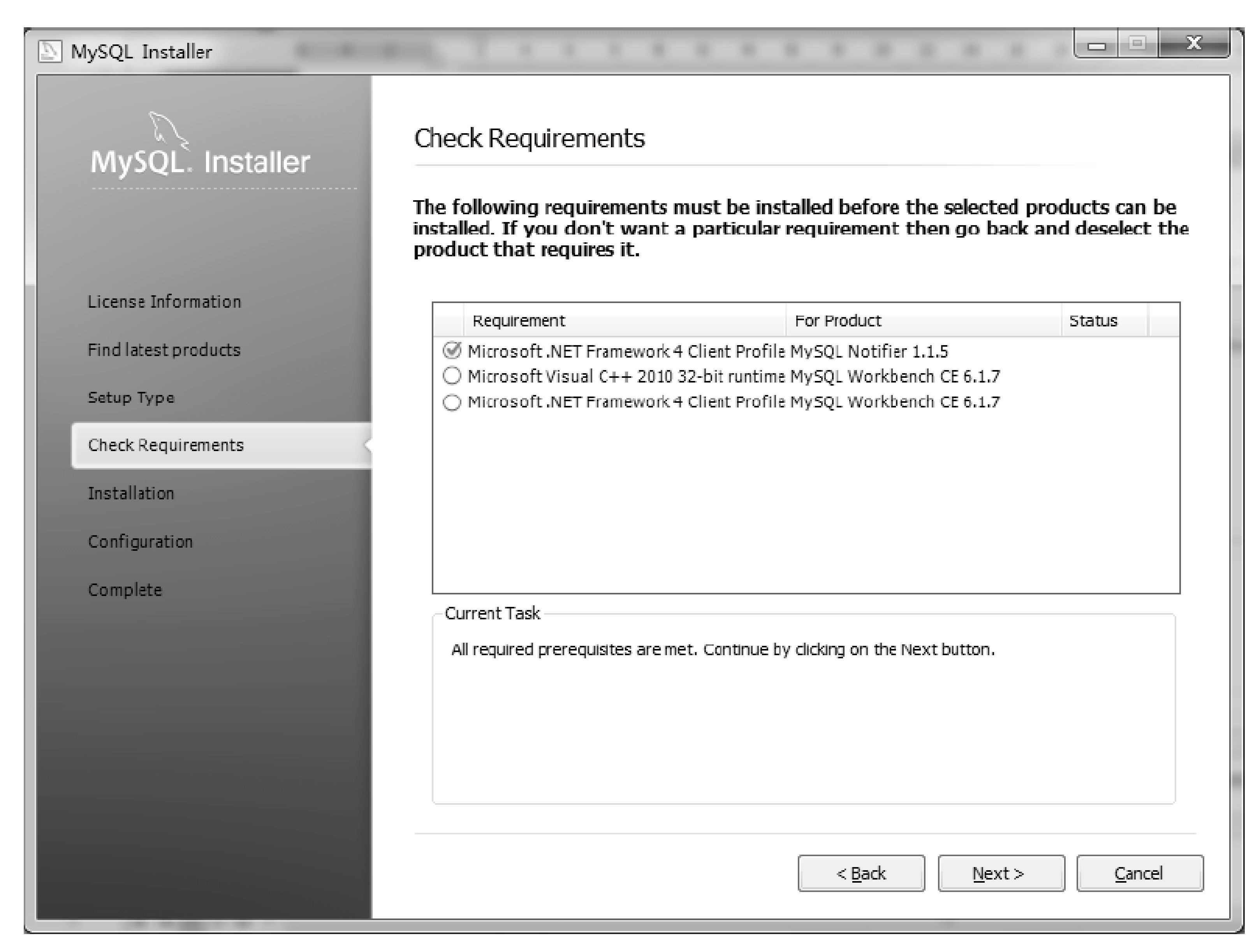


图 2.14　未满足全部安装条件时的 Check Requirements 对话框

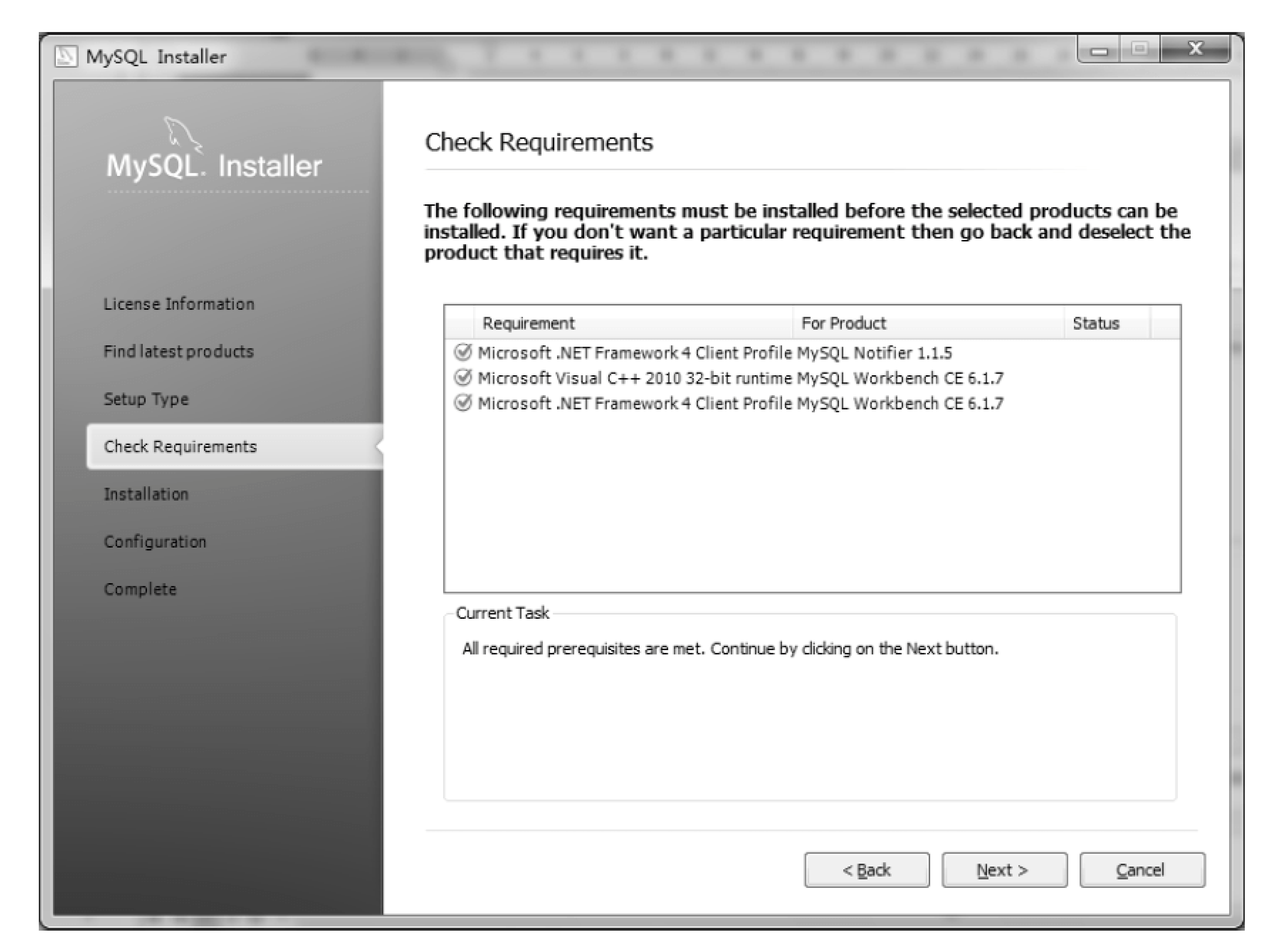


图 2.15　安装条件已全部满足时的 Check Requirements 对话框

（6）单击 Next 按钮，将打开如图 2.16 所示的 Installation Progress 对话框。

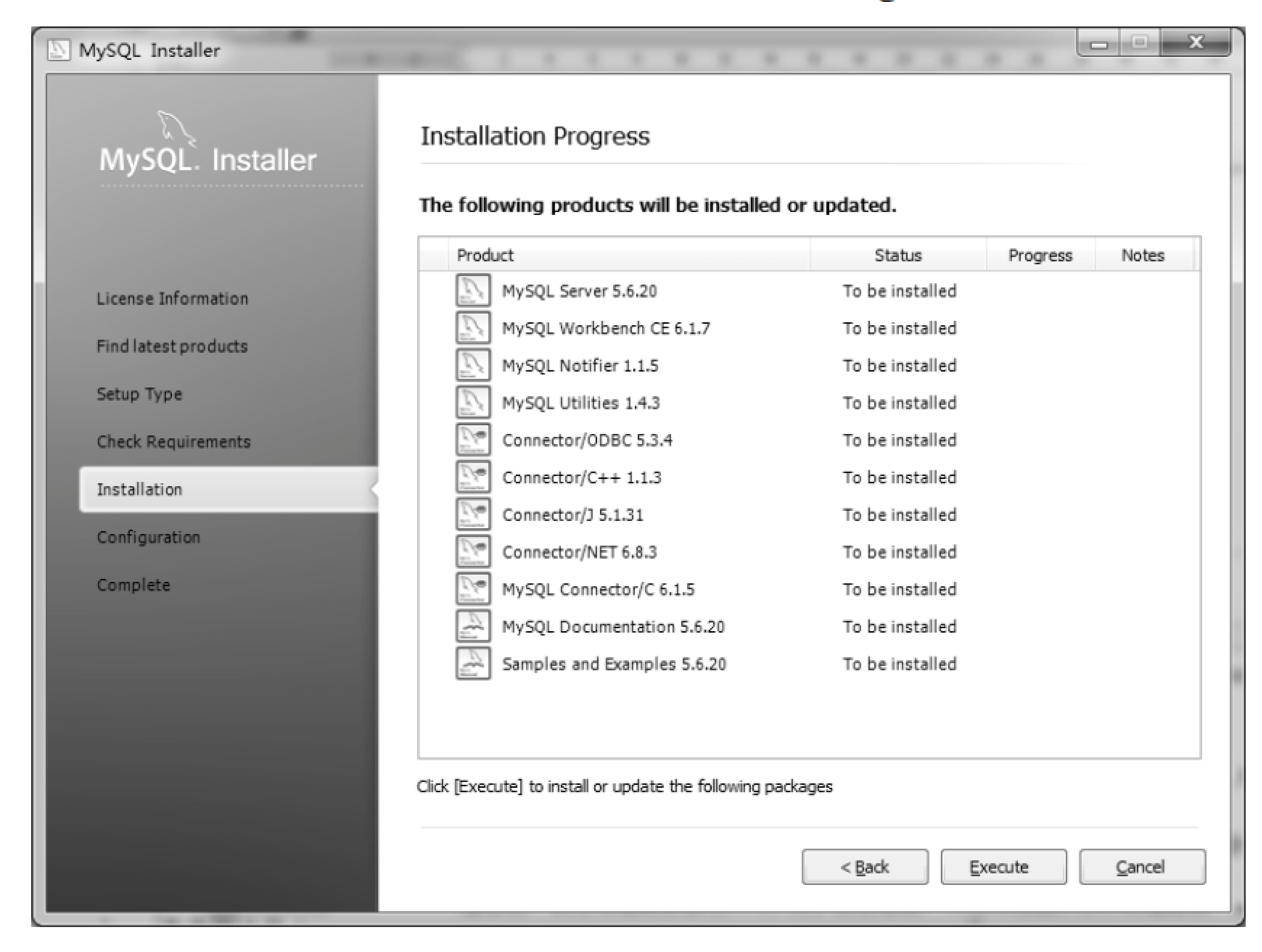


图 2.16　未安装完成的 Installation Progress 对话框

（7）单击 Execute 按钮，将开始安装，并显示安装进度。安装完成后，将显示如图 2.17 所示的对话框。

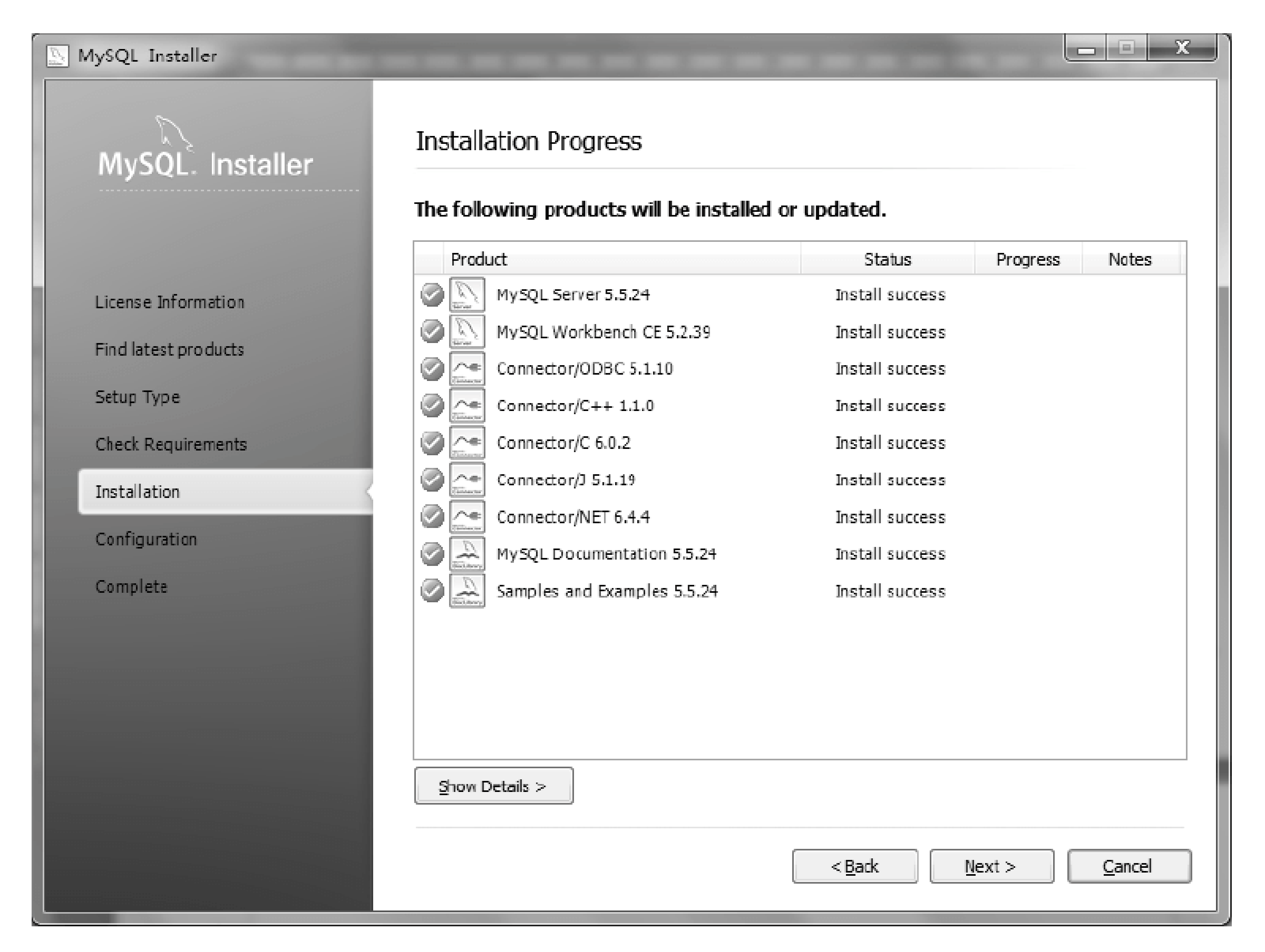


图 2.17　安装完成时的 Installation Progress 对话框

（8）单击 Next 按钮，将打开 Configuration Overview 对话框，在该对话框中单击 Next 按钮，将打开用于选择服务器的类型的 MySQL Server Configuration 对话框，在该对话框中共提供了 Development Machine（开发者类型）、Server Machine（服务器类型）和 Dedicated Machine（致力于 MySQL 服务类型）。这里选择默认的 Development Machine，如图 2.18 所示。



图 2.18　配置服务器类型和网络选项的对话框



MySQL 使用的默认端口是 3306，在安装时，可以修改为其他的（如 3307）。但是一般情况下，不要修改默认的端口号，除非 3306 端口已经被占用。

（9）单击 Next 按钮，将打开用于设置用户和安全的 MySQL Server Configuration 对话框，在这个对话框中，可以设置 root 用户的登录密码，也可以添加新用户，这里只设置 root 用户的登录密码为 root，其他采用默认，如图 2.19 所示。

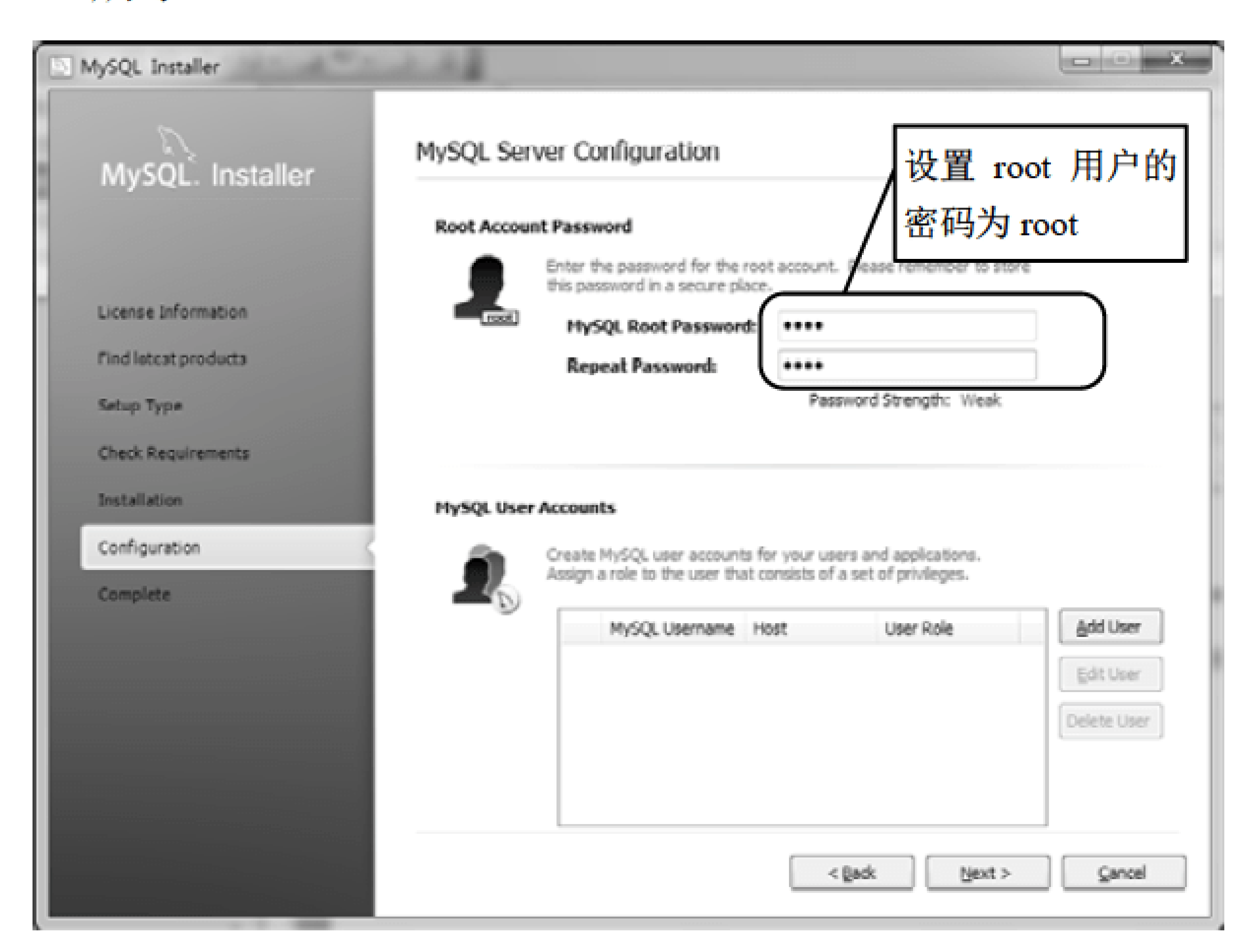


图 2.19　设置用户和安全的 MySQL Server Configuration 对话框

（10）单击 Next 按钮，将打开 Configuration Overview 对话框，开始配置 MySQL 服务器，这里采用默认设置，如图 2.20 所示。

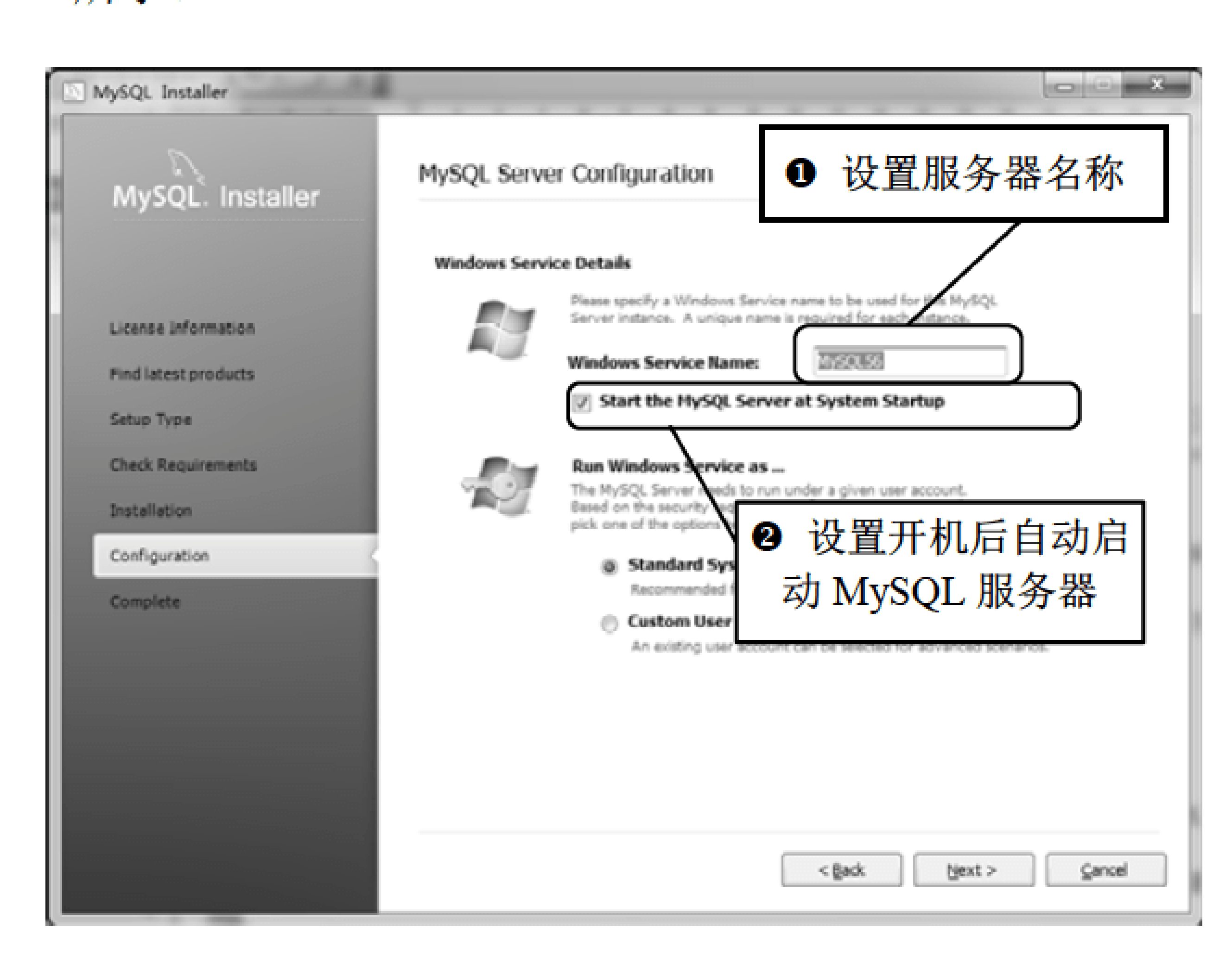


图 2.20　配置 MySQL 服务器

（11）单击 Next 按钮，将显示如图 2.21 所示的界面，提示安装 MySQL 提供的简单示例。

图 2.21　提示安装 MySQL 提供的简单示例

（12）单击 Next 按钮，开始安装，安装完成后，将显示如图 2.22 所示的界面。

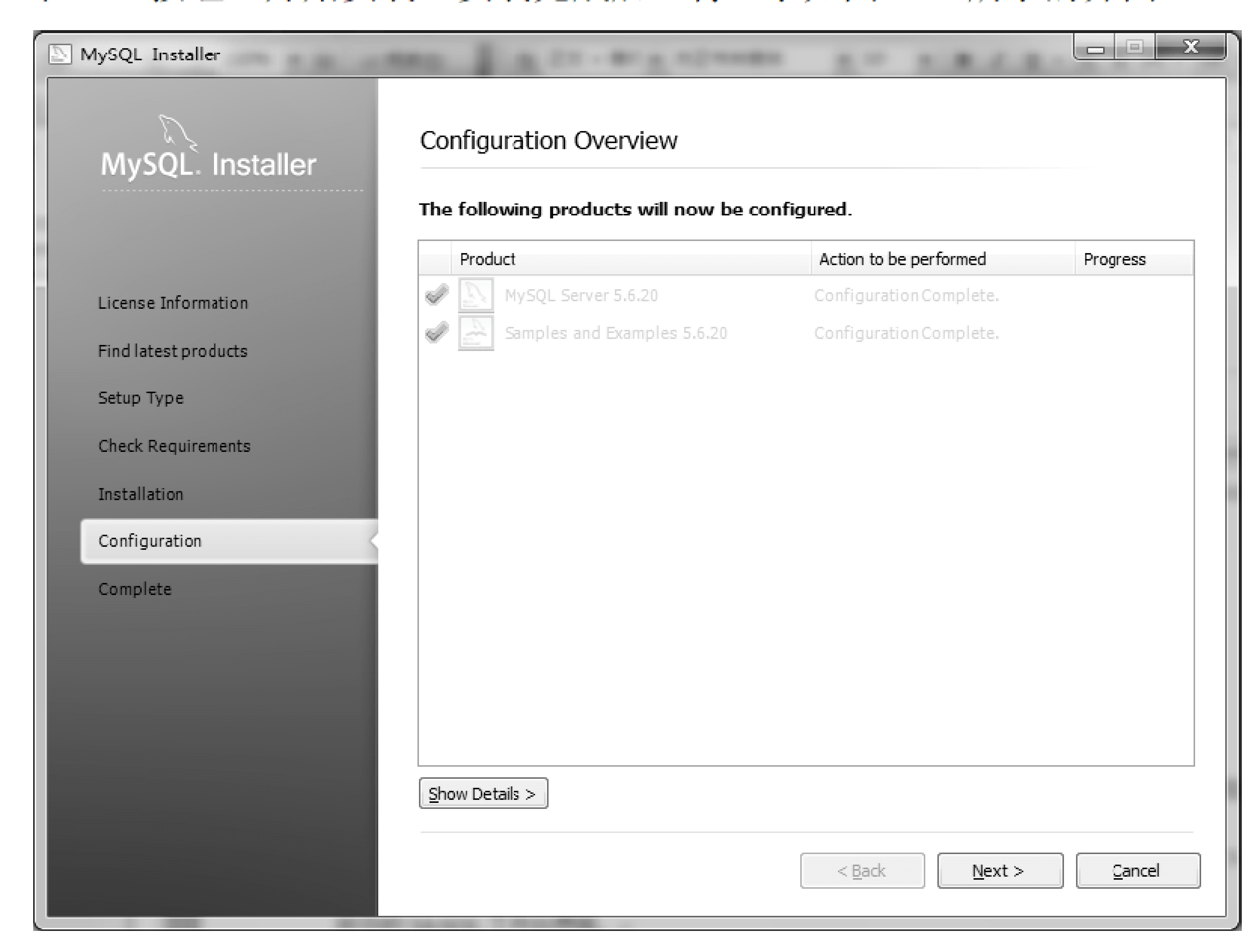


图 2.22　配置完成界面

（13）单击Next按钮，将显示如图2.23所示的安装完成界面。取消选中Start MySQL Workbench after Setup 复选框，单击 Finish 按钮，完成 MySQL 的安装。

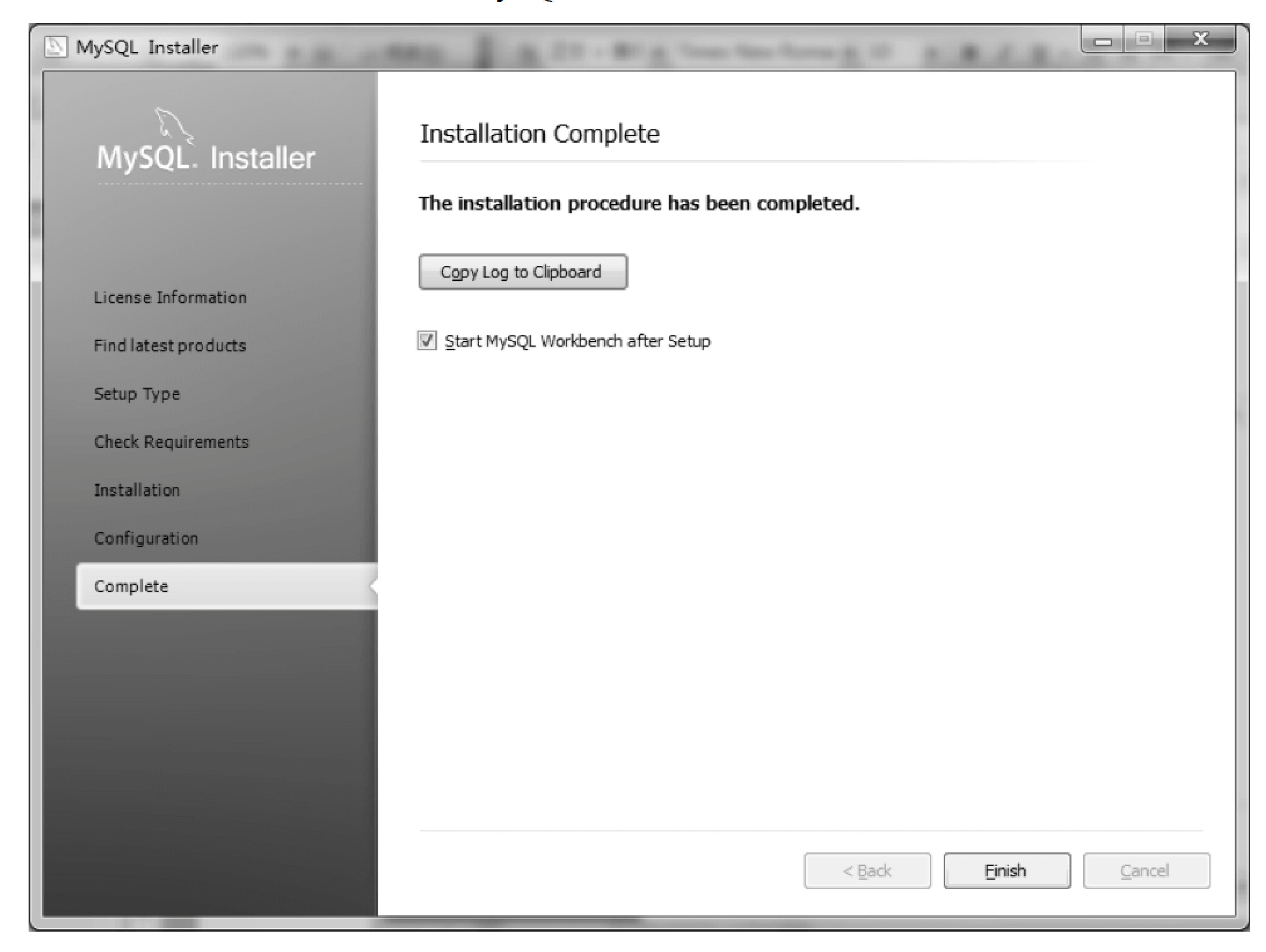


图 2.23　安装完成对话框

2.4.3　启动、连接、断开和停止 MySQL 服务器

通过系统服务器和命令提示符（DOS）都可以启动、连接断开和停击 MySQL，操作非常简单。下面以 Windows 7 操作系统为例，讲解其具体的操作流程。通常情况下不要停止 MySQL 服务器，否则数据库将无法使用。

1. 启动、停止 MySQL 服务器

启动、停止 MySQL 服务器的方法有两种：系统服务器和命令提示符（DOS）。

1）通过系统服务器启动、停止 MySQL 服务器

如果 MySQL 设置为 Windows 服务，则可以通过选择“开始”→“控制面板”→“系统和安全”→“管理工具”→“服务”命令打开 Windows 服务管理器。在服务器的列表中找到 MySQL 服务并右键单击，在弹出的快捷菜单中，完成 MySQL 服务的各种操作（启动、重新启动、停止、暂停和恢复），如图 2.24 所示。



图 2.24　通过系统服务启动、停止 MySQL 服务器

2）在命令提示符下启动、停止 MySQL 服务器

单击“开始”菜单，在出现的命令输入框中输入 cmd 命令，按 Enter 键打开 DOS 窗口。在命令提示符下输入：

```
\> net start mysql
```

此时再按 Enter 键，启用 MySQL 服务器。

在命令提示符下输入：

```
\> net stop mysql
```

按 Enter 键，即可停止 MySQL 服务器。在命令提示符下启动、停止 MySQL 服务器的运行效果如图 2.25 所示。



图 2.25　在命令提示符下启动、停止 MySQL 服务器

2. 连接和断开 MySQL 服务器

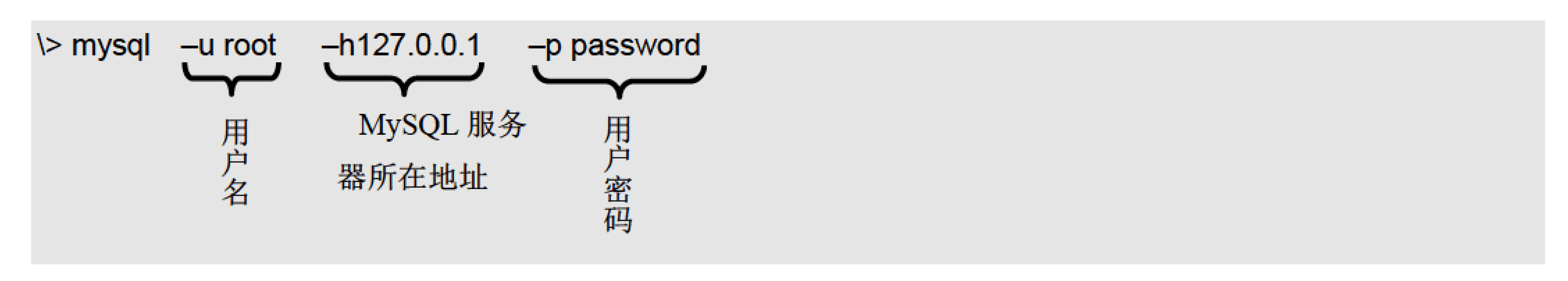
下面分别介绍连接和断开 MySQL 服务器的方法。

1）连接 MySQL 服务器

连接 MySQL 服务器通过 mysql 命令实现。在 MySQL 服务器启动后，选择“开始”→“运行”命



令，在弹出的“运行”窗口中输入 cmd 命令，按 Enter 键后进入 DOS 窗口，在命令提示符下输入：



注意

在连接 MySQL 服务器时，MySQL 服务器所在地址（如 - h127.0.0.1）可以省略不写。

输入完命令语句后，按 Enter 键即可连接 MySQL 服务器，如图 2.26 所示。



图 2.26　连接 MySQL 服务器

说明

为了保护 MySQL 数据库的密码，可以采用如图 2.26 所示的密码输入方式。如果密码在-p 后直接给出，那么密码就以明文显示，例如：

mysql －u root －h127.0.0.1 －p root

按 Enter 键后再输入密码（以加密的方式显示），然后按 Enter 键即可成功连接 MySQL 服务器。

如果用户在使用 mysql 命令连接 MySQL 服务器时弹出如图 2.27 所示的信息，那么说明用户未设置系统的环境变量。



图 2.27　连接 MySQL 服务器出错

也就是说，没有将 MySQL 服务器的 bin 文件夹位置添加到 Windows 的“环境变量”→“系统变量”→Path 中，从而导致命令不能执行。

下面介绍这个环境变量的设置方法，其步骤如下。

（1）右键单击“计算机”图标，在弹出的快捷菜单中选择“属性”命令，在弹出的对话框中选择“高级系统设置”，弹出“系统属性”对话框，如图 2.28 所示。

（2）在“系统属性”对话框中，选择“高级”选项卡，单击“环境变量”按钮，弹出“环境变量”对话框，如图 2.29 所示。



图 2.28　“系统属性”对话框



图 2.29　“环境变量”对话框

（3）在“环境变量”对话框中，定位到“系统变量”中的 Path 选项，单击“编辑”按钮，将弹出“编辑系统变量”对话框，如图 2.30 所示。



图 2.30　“编辑系统变量”对话框

（4）在“编辑系统变量”对话框中，将 MySQL 服务器的 bin 文件夹位置（C:\Program Files\MySQL\MySQL Server 5.6\bin）添加到“变量值”文本框中，注意要使用“;”与其他变量值进行分隔，最后，单击“确定”按钮。

环境变量设置完成后，再使用 mysql 命令即可成功连接 MySQL 服务器。

2）断开 MySQL 服务器

连接到 MySQL 服务器后，可以通过在 MySQL 提示符下输入 exit 或者 quit 命令断开 MySQL 连接，



格式如下。

```
mysql> quit;
```

2.4.4　打开 MySQL 5.6 Command Line Client

MySQL 服务器安装完成后，就可以通过其提供的 MySQL 5.6 Command Line Client 程序来操作 MySQL 数据了。这时，必须先打开 MySQL 5.6 Command Line Client 程序，并登录 MySQL 服务器。下面将介绍具体的步骤。

（1）在“开始”菜单中，选择“所有程序”→MySQL→MySQL Server 5.6→MySQL 5.6 Command Line Client 命令，将打开 MySQL 5.6 Command Line Client 窗口，如图 2.31 所示。



图 2.31　MySQL 客户端命令行窗口

（2）在该窗口中，输入 root 用户的密码（这里为 root），将登录到 MySQL 服务器，如图 2.32 所示。

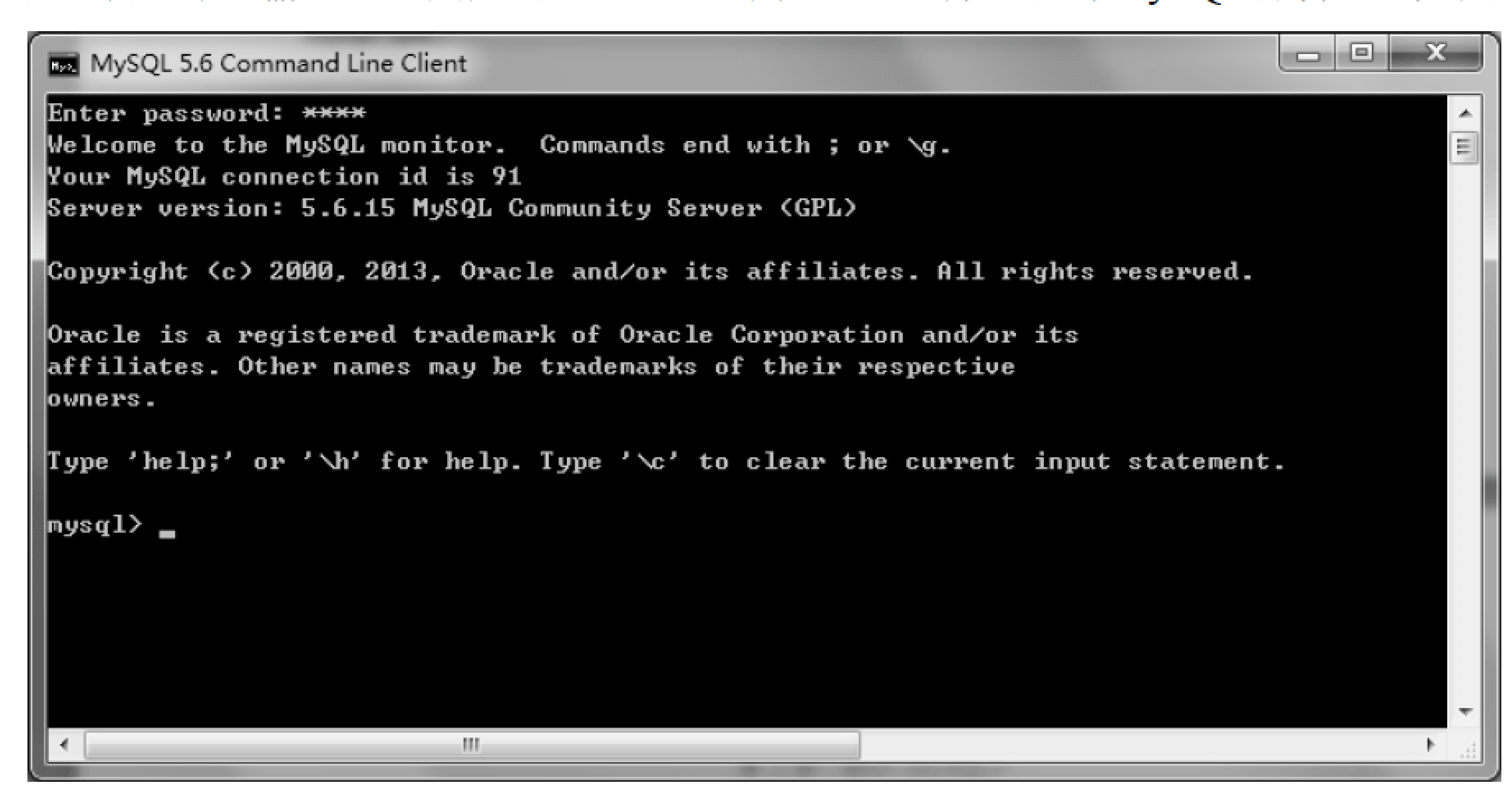


图 2.32　登录到 MySQL 服务器

2.5　如何学好 MySQL

学好 MySQL 最重要的是要多练习。笔者将自己学习数据库的方法总结如下。

1. 多上机实践

要想熟练地掌握数据库，必须经常上机练习。只有在上机实践中才能深刻体会数据库的使用。通常情况下，数据库管理员工作的时间越长，其工作经验就越丰富。很多复杂的问题，都可以根据数据库管理员的经验来更好地解决。上机实践的过程中，可以将学到的数据库理论知识理解得更加透彻。

2. 多编写 SQL 语句

SQL 语句是数据库的灵魂。数据库中的很多操作都是通过 SQL 语句来实现的。只有经常使用 SQL 语句来操作数据库中的数据，读者才可以更加深刻地理解数据库。

3. 数据库理论知识不能丢

数据库理论知识是学好数据库的基础。虽然理论知识会有点儿枯燥，但是这是学好数据库的前提。例如，数据库理论中会涉及 E-R 图、数据库设计原则等知识。如果不了解这些知识，就很难独立设计一个很好的数据库及表。读者可以将数据库理论知识与上机实践结合到一起来学习，这样效率会提高。

2.6　小　　结

本章介绍了数据库和 MySQL 的基础知识。通过本章的学习，希望读者对数据库、MySQL 数据库和 SQL 等知识有所了解。而且，希望读者能够了解常用的数据库系统。第 3 章将介绍在 Windows 操作系统下安装和配置 MySQL，同时对数据库的相关知识也要有一定的了解。

2.7　实践与练习

1. 到 Oracle 的官网中下载最新版本的 MySQL 服务器并安装。
2. 尝试在命令提示符下启动、停止 MySQL 服务器。

第 3 章

使用 MySQL 图形化管理工具

（视频讲解：25 分钟）

MySQL 的管理维护工具非常多，除了系统自带的命令行管理工具之外，还有许多其他的图形化管理工具，常用的有 MySQL Workbench、phpMyAdmin、Navicat 等。通过这些第三方的管理工具，可以使 MySQL 的管理更加方便。本章将对 MySQL Workbench 图形化管理工具和 phpMyAdmin 图形化管理工具进行系统讲解。

通过阅读本章，读者可以：

- 了解 MySQL Workbench 图形化管理工具
- 通过 MySQL Workbench 图形化管理工具创建数据库和数据表
- 通过 MySQL Workbench 图形化管理工具添加数据
- 通过 MySQL Workbench 图形化管理工具导出导入数据
- 配置 phpMyAdmin 图形化管理工具
- 使用 phpMyAdmin 图形化管理工具实现数据库与数据表的创建
- 通过 phpMyAdmin 图形化管理工具对数据库与数据表进行管理
- 使用 phpMyAdmin 图形化管理工具对数据库执行导入导出
- 通过 phpMyAdmin 图形化管理工具设置数据的编码格式
- 通过 phpMyAdmin 图形化管理工具添加服务器新用户
- 通过 phpMyAdmin 图形化管理工具重置 MySQL 服务器用户密码

3.1 MySQL Workbench 图形化管理工具

MySQL Workbench 是 MySQL AB 发布的可视化的数据库设计软件，它的前身是 FabForce 公司的 DB Designer 4。MySQL Workbench 是为数据库管理员、程序开发者和系统规划师设计的统一的可视化工具。它提供了先进的数据建模，灵活的 SQL 编辑器和全面的管理工具，可在 Windows、Linux 和 Mac 等操作系统上使用。

数据建模——MySQL Workbench 包括所有数据建模工程需要的功能，能正向和反向建立复杂的 ER 模型，也提供了通常需要花更多时间才能完成的变更管理和文档任务的关键功能。

SQL 编辑器——MySQL Workbench 提供了用于创建、执行和优化 SQL 查询的可视化工具。SQL 编辑器提供了语法高亮显示，SQL 代码复用和执行的 SQL 历史。数据库的连接面板允许开发人员轻松地管理数据库连接。对象浏览器提供即时访问数据库模型和对象。

管理工具——MySQL Workbench 提供了可视化的控制台，能轻松管理 MySQL 数据库环境，并为数据库增加了更好的可视性。开发人员和 DBA 可以使用可视化工具配置服务器，管理用户和监控数据库的健康状况。

3.1.1 了解 MySQL Workbench

MySQL 数据库安装完成后，将自动安装一个图形化工具，用于创建并管理数据库。在“开始”菜单中选择“所有程序”→MySQL→MySQL Workbench 6.1 CE 命令，将打开如图 3.1 所示的 MySQL Workbench 主界面。

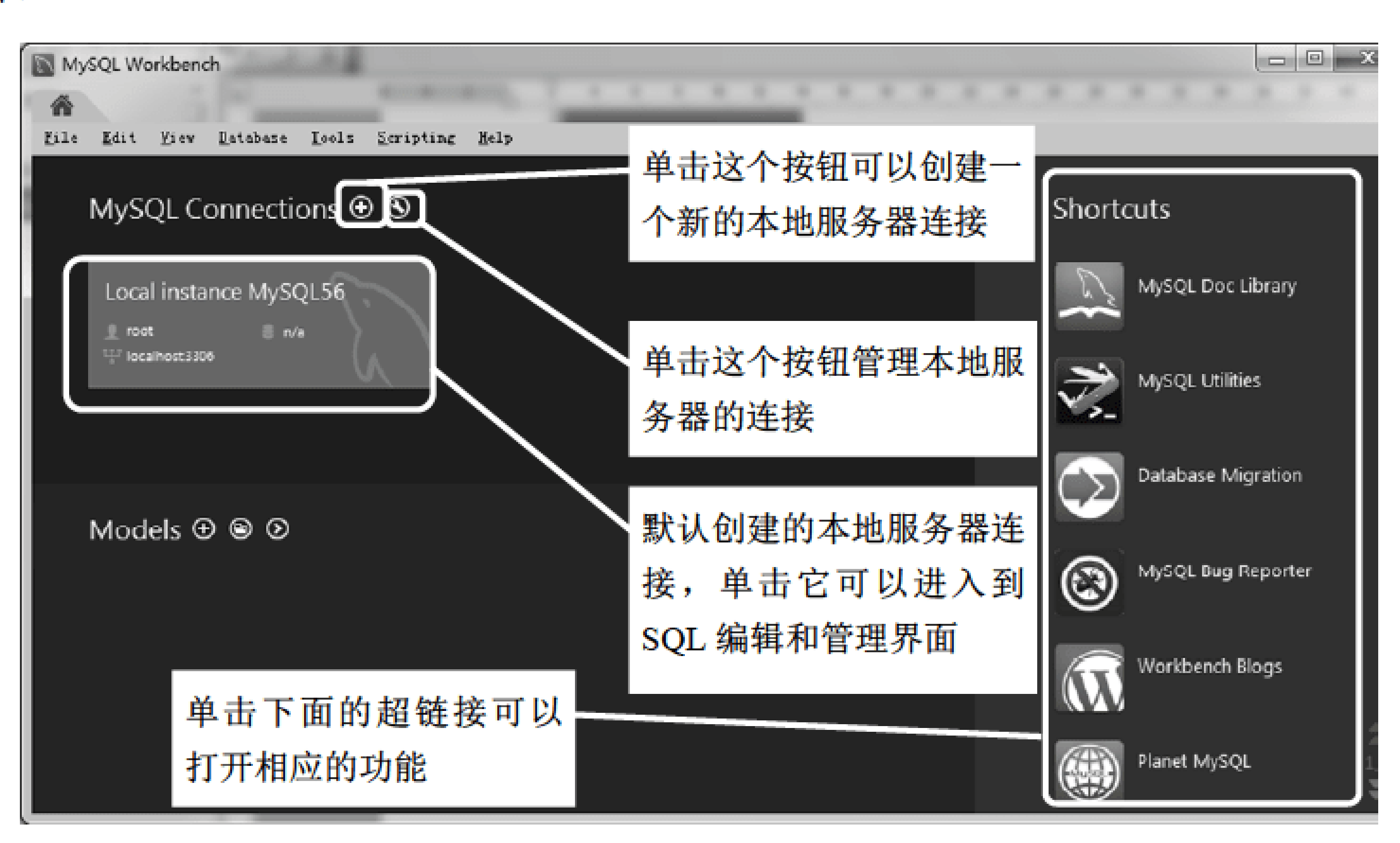


图 3.1 MySQL Workbench 主界面

在图 3.1 中，单击 Local instance MySQL56 超链接，将打开一个输入用户密码的对话框，在该对话框中输入 root 用户的密码，这里为 root，如图 3.2 所示。





图 3.2　输入用户密码对话框

单击 OK 按钮，将打开如图 3.3 所示的 Local instance MySQL56 选项卡。在该选项卡中，可以进行创建/管理数据库、创建/管理数据表、编辑表数据和查询表数据等操作。

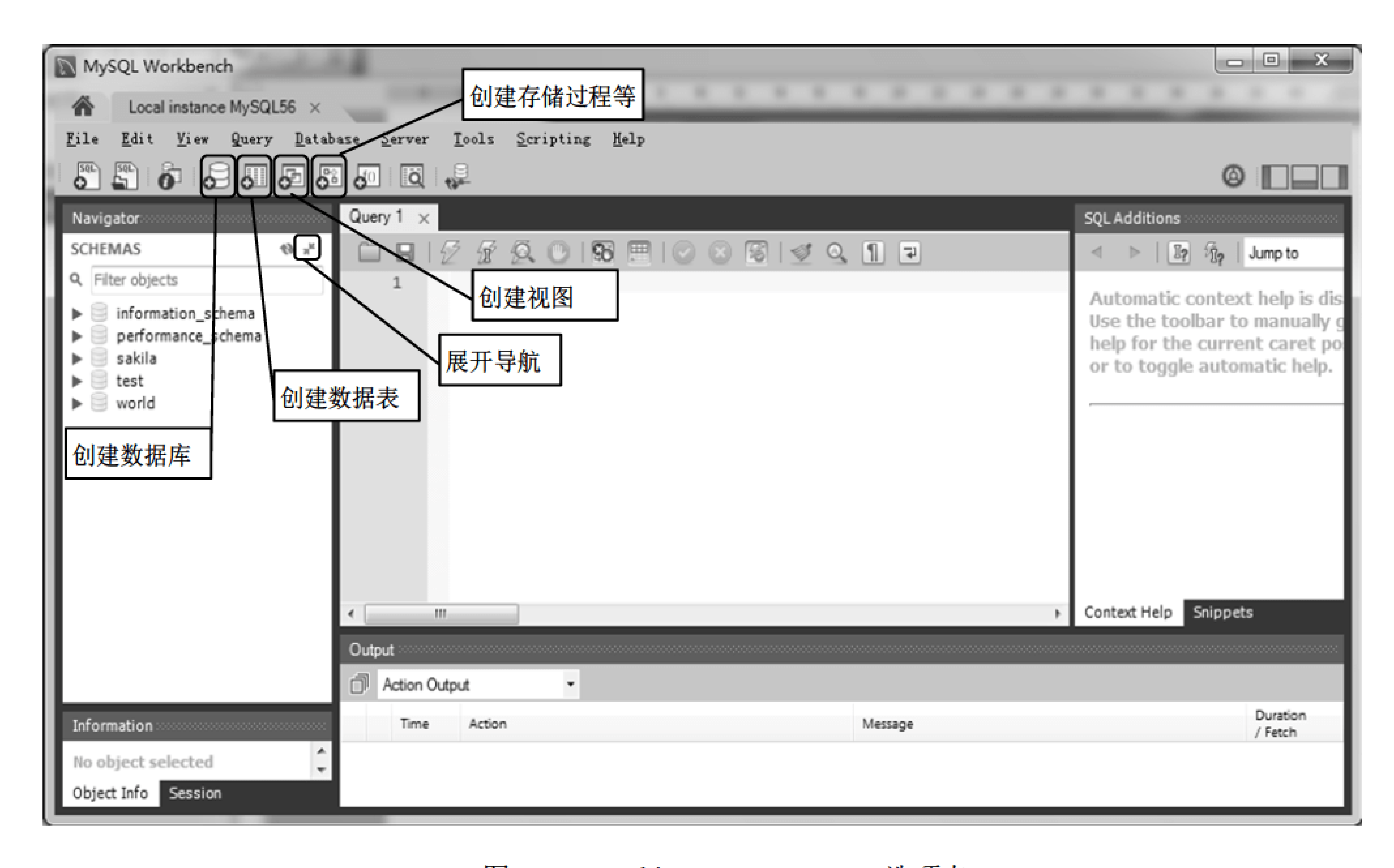


图 3.3　Local instance MySQL56 选项卡

3.1.2　创建数据库和数据表

下面将介绍如何应用 MySQL Workbench 图形化管理工具创建数据库和数据表。

1．创建数据库

通过 MySQL Workbench 创建数据库的具体方法如下。

（1）打开 MySQL Workbench 的 Local instance MySQL56 选项卡，单击工具栏中的“创建数据库”

按钮，将打开如图 3.4 所示的新建数据库选项卡。

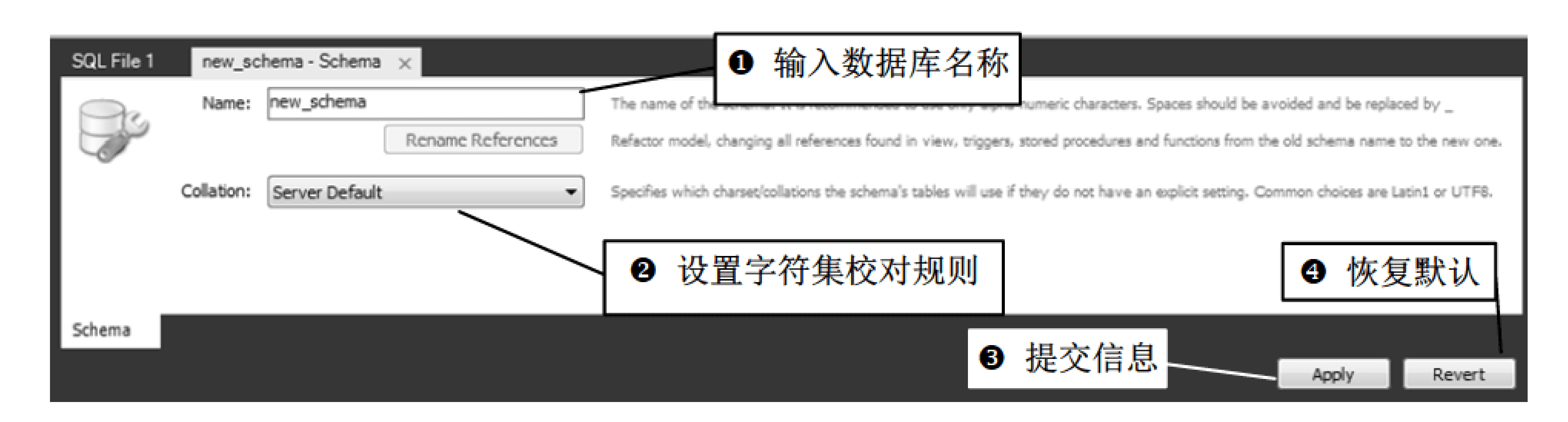


图 3.4　新建数据库选项卡



在 MySQL 中，Create Schema 和 Create Database 的作用是一样的，都是创建数据库。在 MySQL Workbench 中，创建数据库使用的是 Create Schema。

（2）在如图 3.4 所示的新建数据库选项卡中，在 Name 文本框中输入数据库名称。这里的数据库名称必须符合操作系统文件夹的命名规则，而在 MySQL 中是不区分大小写的。这里创建一个名称为 db_database02 的数据库。

（3）在 Collation 下拉列表框中选择所创建数据库的字符集校对规则。例如，要使用 UTF8 编码，则可以选择 utf8 - utf8_general_ci，也可以选择 utf8 - default collation。这里选择 utf8 - default collation，如图 3.5 所示。

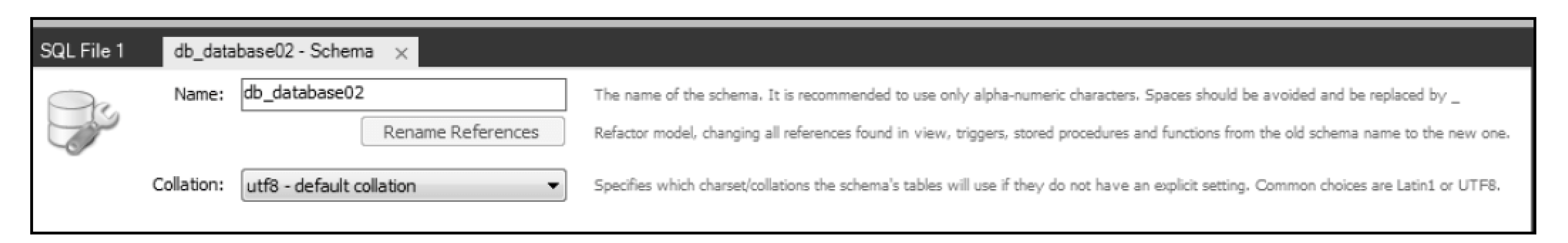


图 3.5　创建数据库 db_database02

（4）单击 Apply 按钮，将弹出如图 3.6 所示的对话框，显示生成的可编辑的创建数据库的 SQL 语句。

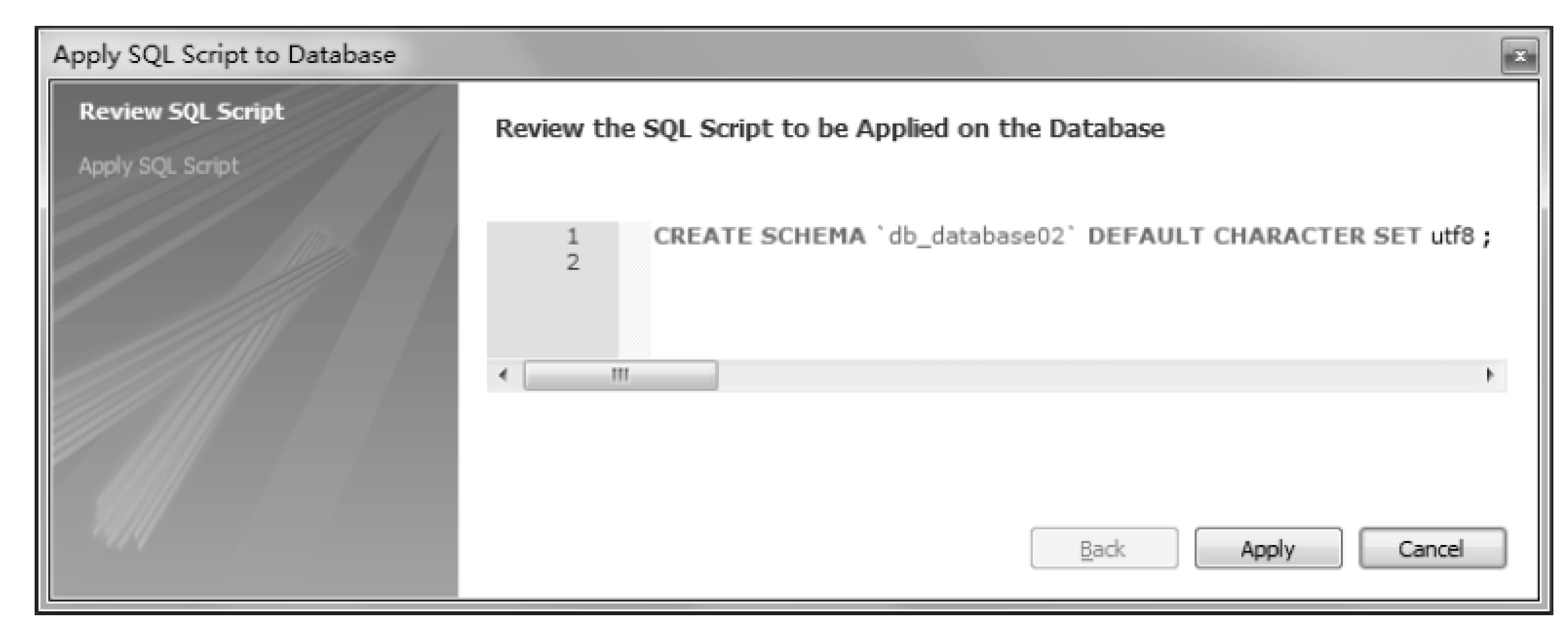


图 3.6　可编辑的创建数据库的 SQL 语句对话框



（5）单击 Apply 按钮，开始创建数据库。数据库创建完成后，显示如图 3.7 所示的完成对话框。

（6）单击 Finish 按钮，关闭完成对话框。

创建数据库后，在 Local instance MySQL56 选项卡的左侧的 SCHEMAS 列表中，将显示新创建的数据库，如图 3.8 所示。





图 3.7　创建数据库完成对话框　　　　图 3.8　新创建的数据库

2. 创建数据表

通常情况下，创建数据库后，还需要创建数据表。在 MySQL Workbench 中，创建数据表前，需要先选择要创建数据表的那个数据库，这时可以在 Local instance MySQL56 选项卡的左侧的 SCHEMAS 列表中，双击该数据表来将其设置为默认数据库，然后就可创建数据表了。通过 MySQL Workbench 创建数据表的具体方法如下。

（1）打开 MySQL Workbench 的 Local instance MySQL56 选项卡，单击工具栏中的“创建数据表”按钮，将打开如图 3.9 所示的新建表选项卡。

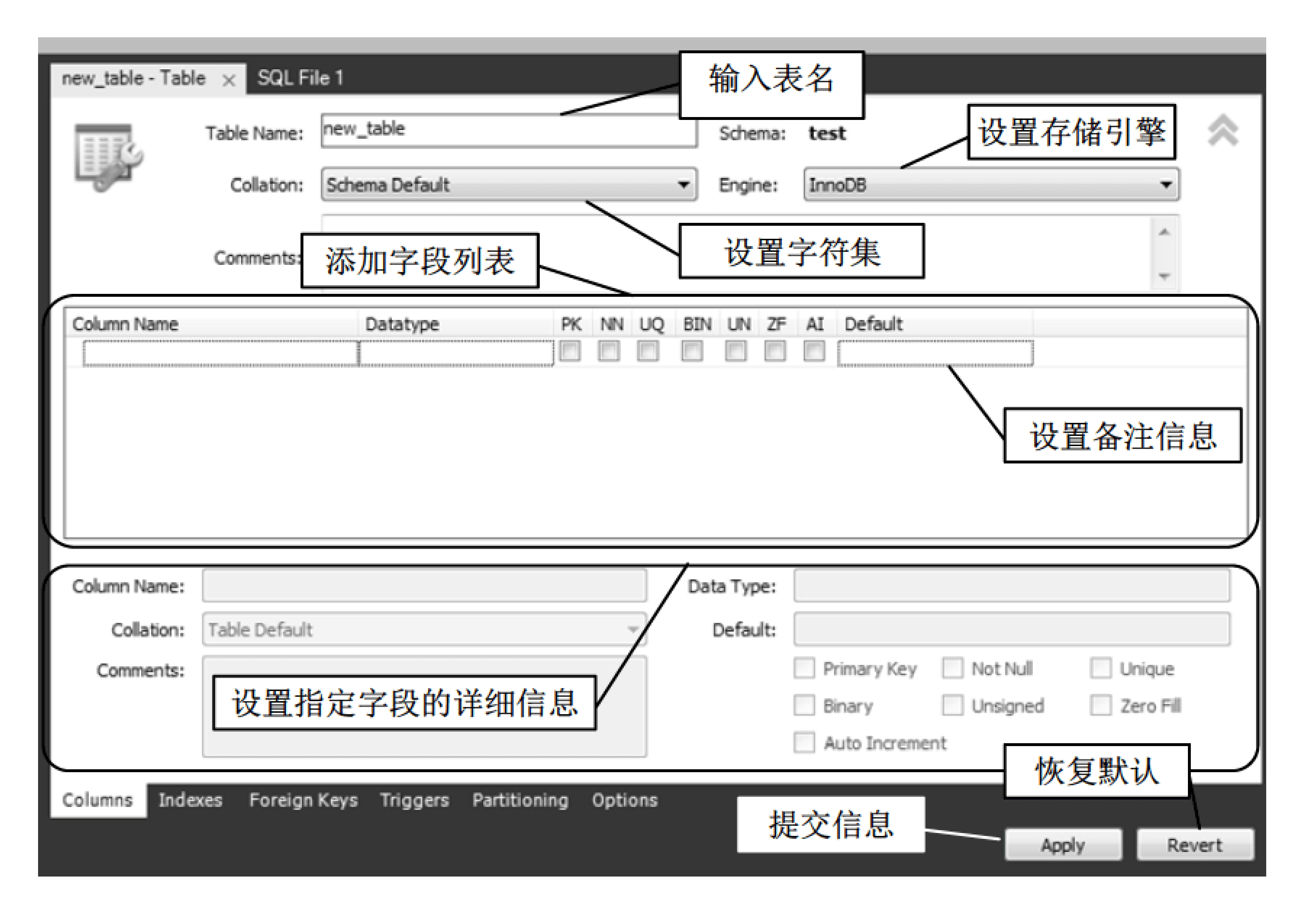


图 3.9　新建表选项卡

（2）在新建表选项卡的 Table Name 文本框中，输入数据表名称（这里为 tb_user）；在 Collation 下拉列表框中选择字符集较对规则为 utf8 - default collation；在 Engine 下拉列表框中选择存储引擎为 MyISAM，如图 3.10 所示。

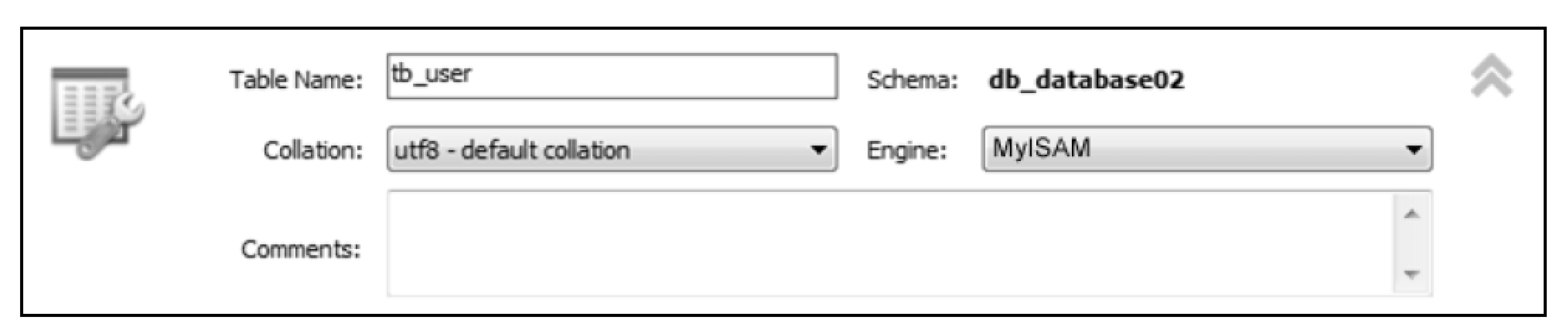


图 3.10　指定数据表名称、字符集校对规则以及存储引擎

（3）在如图 3.9 所示的添加字段列表中，添加如图 3.11 所示的 3 条字段信息。

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	Default
id	INT	☑	☑	☐	☐	☐	☐	☑	
name	VARCHAR(45)	☐	☑	☐	☐	☐	☐	☐	
pwd	VARCHAR(45)	☐	☑	☐	☐	☐	☐	☐	
		☐	☐	☐	☐	☐	☐	☐	

图 3.11　添加字段列表

（4）单击图 3.9 中的 Apply 按钮，将弹出如图 3.12 所示的对话框，显示生成的可编辑的创建数据表的 SQL 语句。

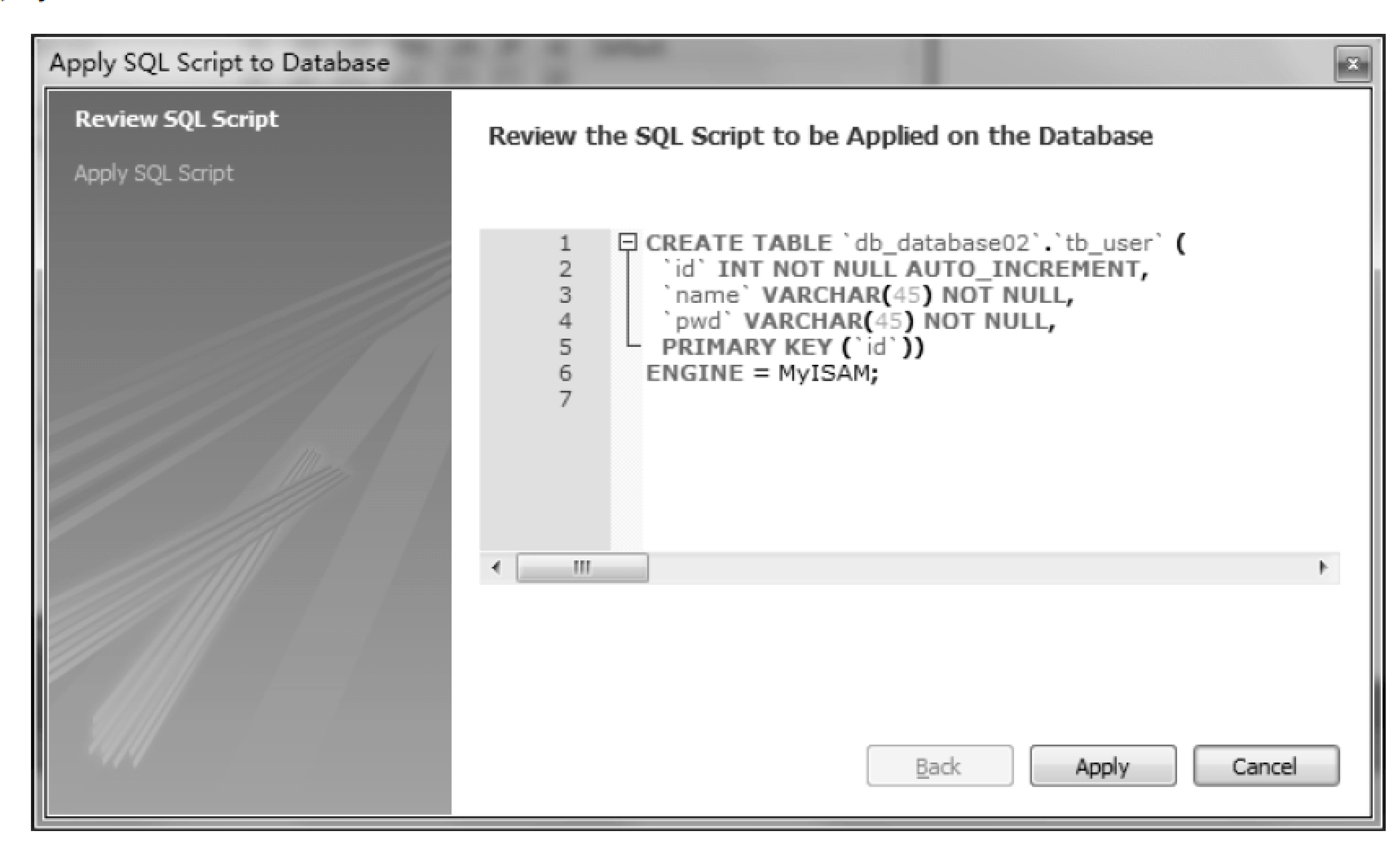


图 3.12　可编辑的创建数据表的 SQL 语句

（5）单击 Apply 按钮，开始创建数据表。数据表创建完成后，显示如图 3.13 所示的完成对话框。



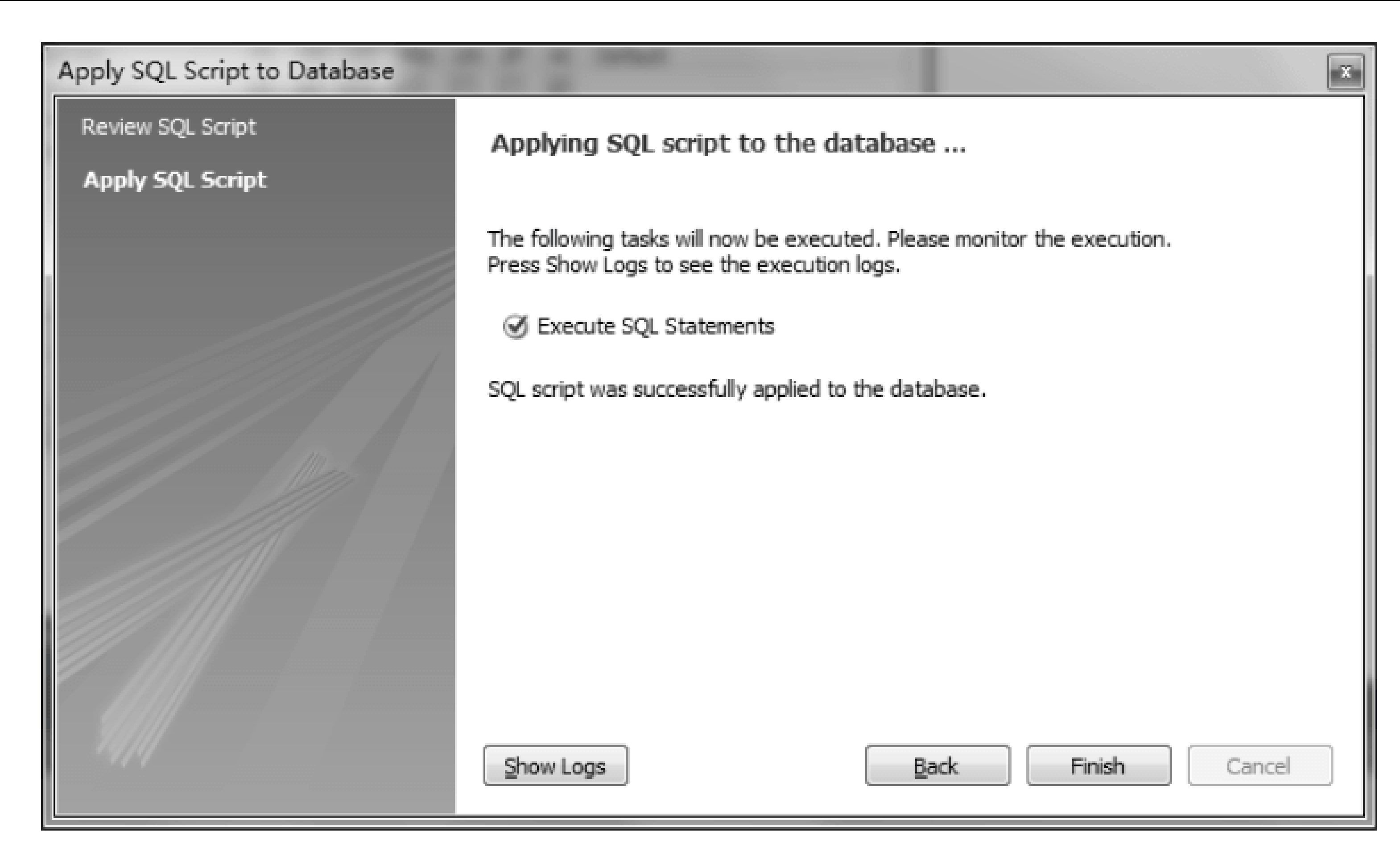


图 3.13　创建数据表完成对话框

（6）单击 Finish 按钮，关闭完成对话框。

3.1.3　添加数据

数据库和数据表创建成功后，就可以向数据表中添加数据了。在 MySQL Workbench 的 Local instance MySQL56 选项卡中，向已有数据表中添加数据的具体步骤如下。

（1）在 Local instance MySQL56 选项卡的左侧的 SCHEMAS 列表中，找到要添加数据的数据表节点，这里为 tb_user 节点，并且在该节点上单击鼠标右键，在弹出的快捷菜单中选择 Select Rows - Limit 5000 命令，如图 3.14 所示。

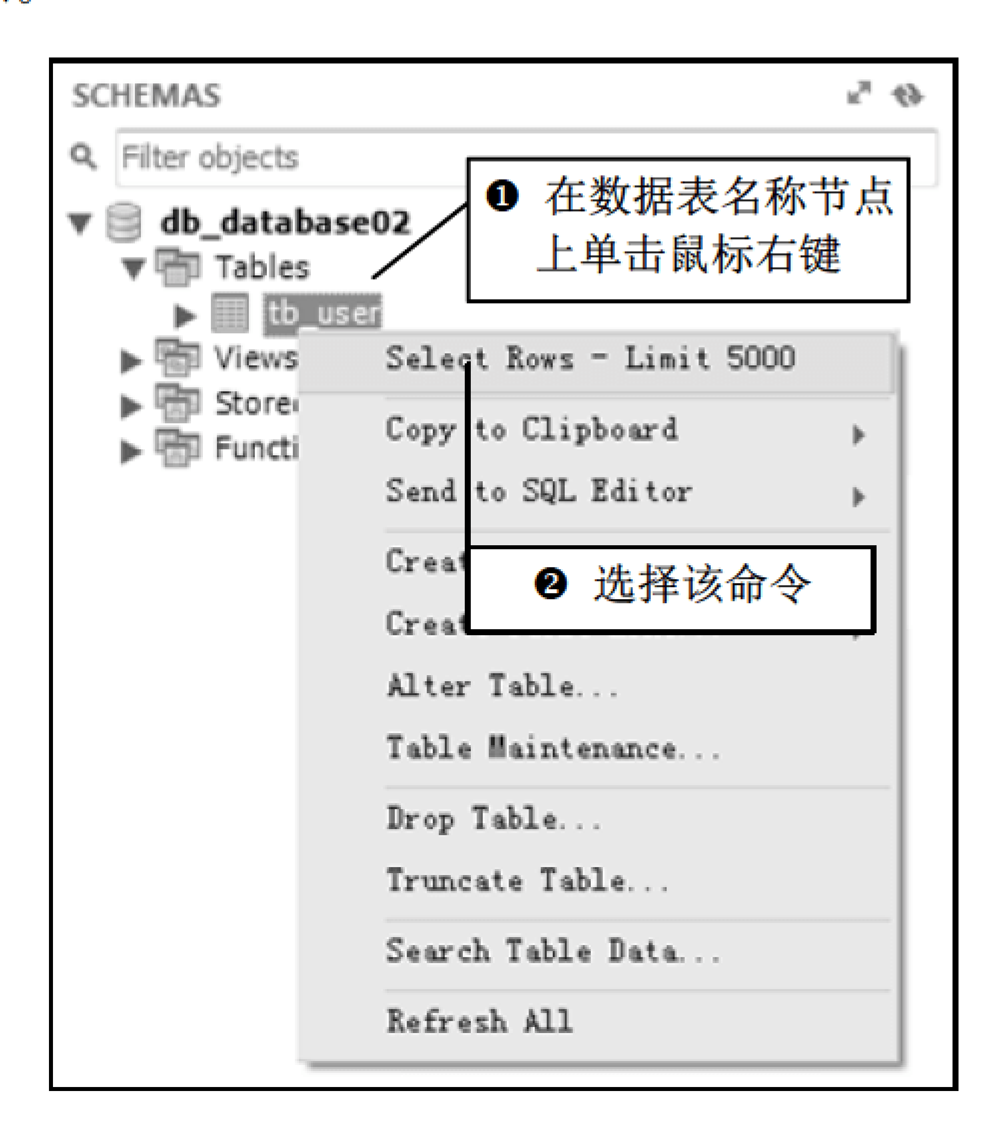


图 3.14　找到要添加数据的数据表 tb_user

技巧

在如图 3.14 所示的快捷菜单中，不但可以执行数据的添加命令，还可以执行查询数据、复制数据、修改表结构和删除表格等操作。

（2）这时将打开一个名为 tb_user 的选项卡，在该选项卡中，上半部分显示的是查询全部数据的 SQL 语句，下半部分是以表格的形式显示的数据表中的数据。由于这个数据表是新创建的，还没有添加任何数据，所以下面的表格是空的，如图 3.15 所示。

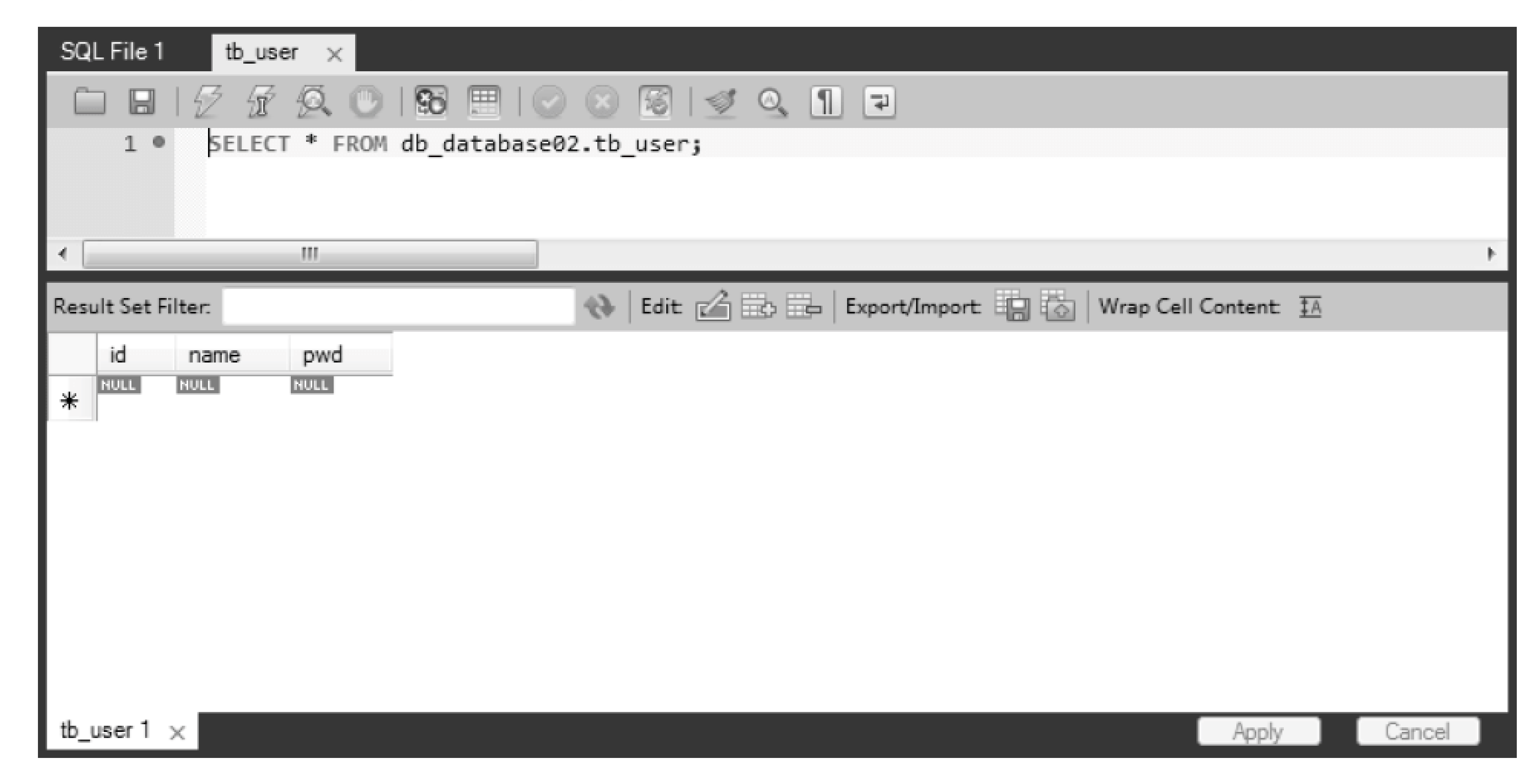


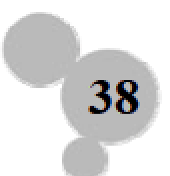
图 3.15　tb_user 选项卡

（3）直接在下面的表格中添加需要的数据，例如，指定用户名为 mr，密码为 111，则可以按如图 3.16 所示的样式进行添加。



图 3.16　向数据表中添加数据

（4）单击图 3.16 中的 Apply 按钮，将弹出如图 3.17 所示的对话框，显示生成的可编辑的添加数据的 SQL 语句。



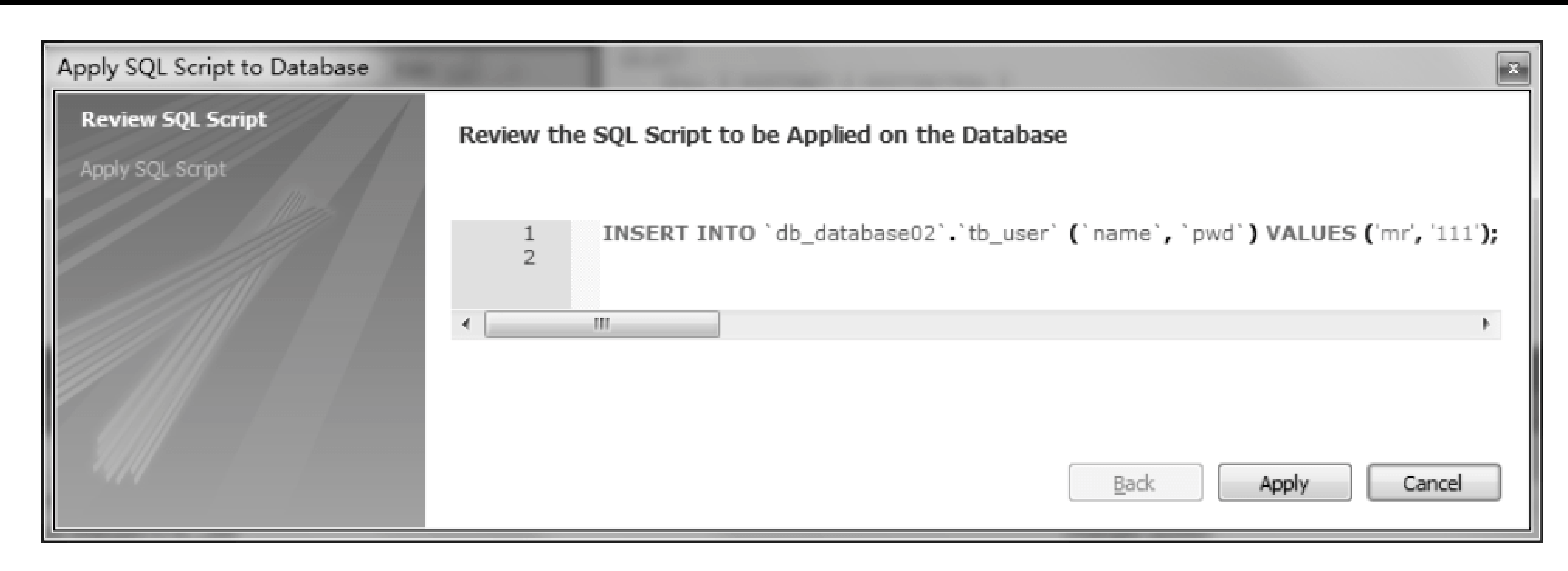


图 3.17　可编辑的添加数据的 SQL 语句

（5）单击 Apply 按钮，开始添加数据。数据添加完成后，显示如图 3.18 所示的完成对话框。

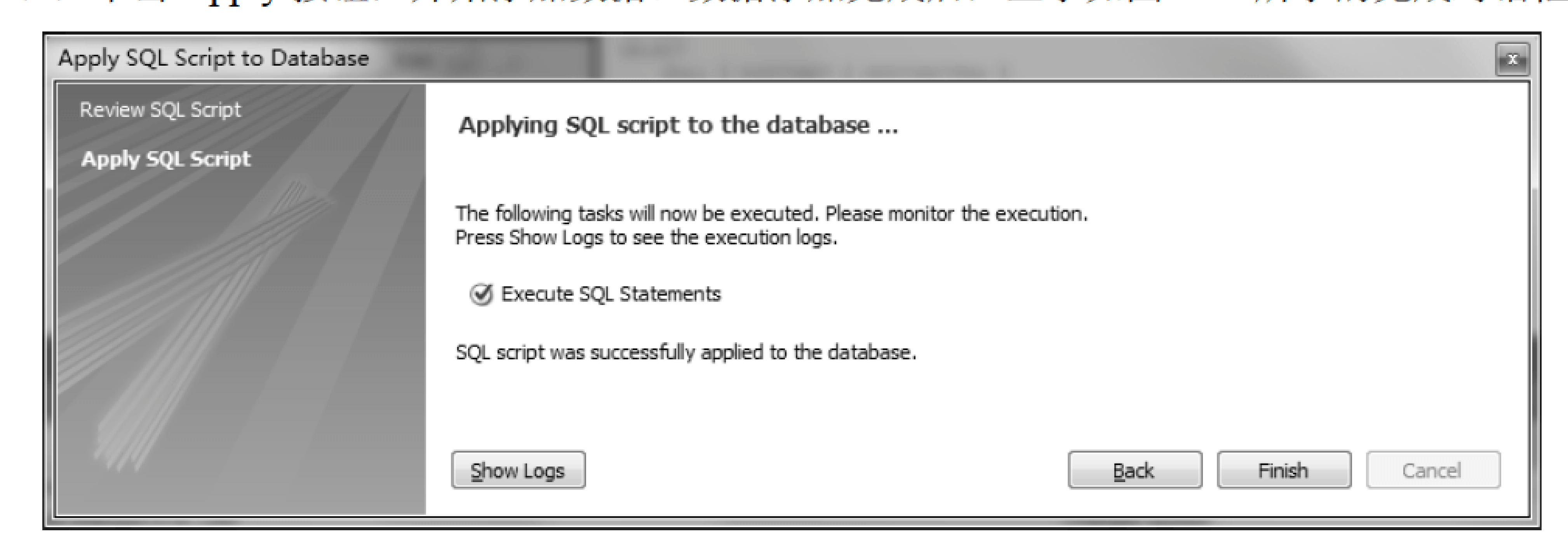


图 3.18　添加数据完成对话框

（6）浏览添加成功后的数据，如图 3.19 所示。



图 3.19　添加成功后的数据

3.1.4　数据的导出和导入

在 MySQL Workbench 的 Local instance MySQL56 选项卡中，单击左侧的“展开导航”图标，可以显示一些常用的导航超链接，其中最上面的 MANAGEMENT 栏目中，提供了 Data Export 和 Data Import/Restore 两个超链接，如图 3.20 所示。

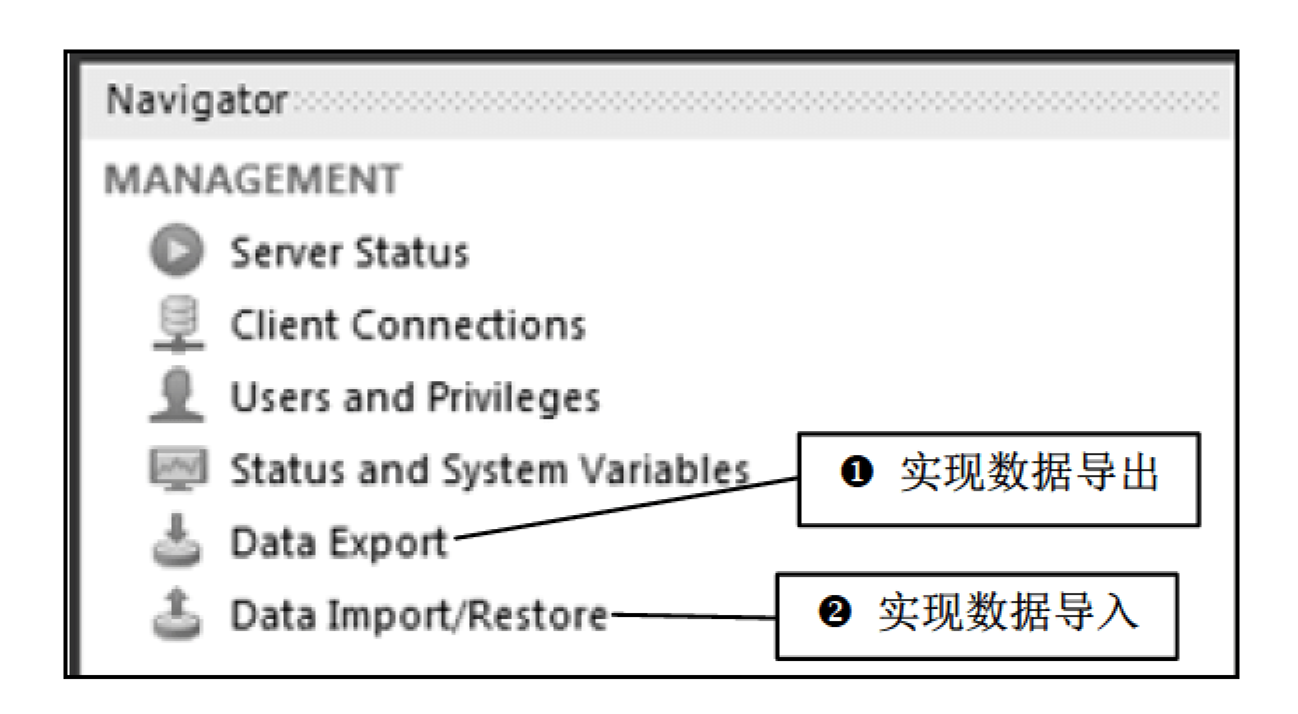


图 3.20　MANAGEMENT 栏目

通过这两个超链接就可以实现数据的导出和导入功能了，下面分别进行介绍。

1．数据导出

导出数据库中数据的具体步骤如下。

（1）在 MySQL Workbench 的 Local instance MySQL56 选项卡的左侧的 MANAGEMENT 栏目中单击 Data Export 超链接，将弹出输入 root 用户密码的对话框，在该对话框中输入密码“root”，单击 OK 按钮，即可显示如图 3.21 所示的 Data Export 选项卡。

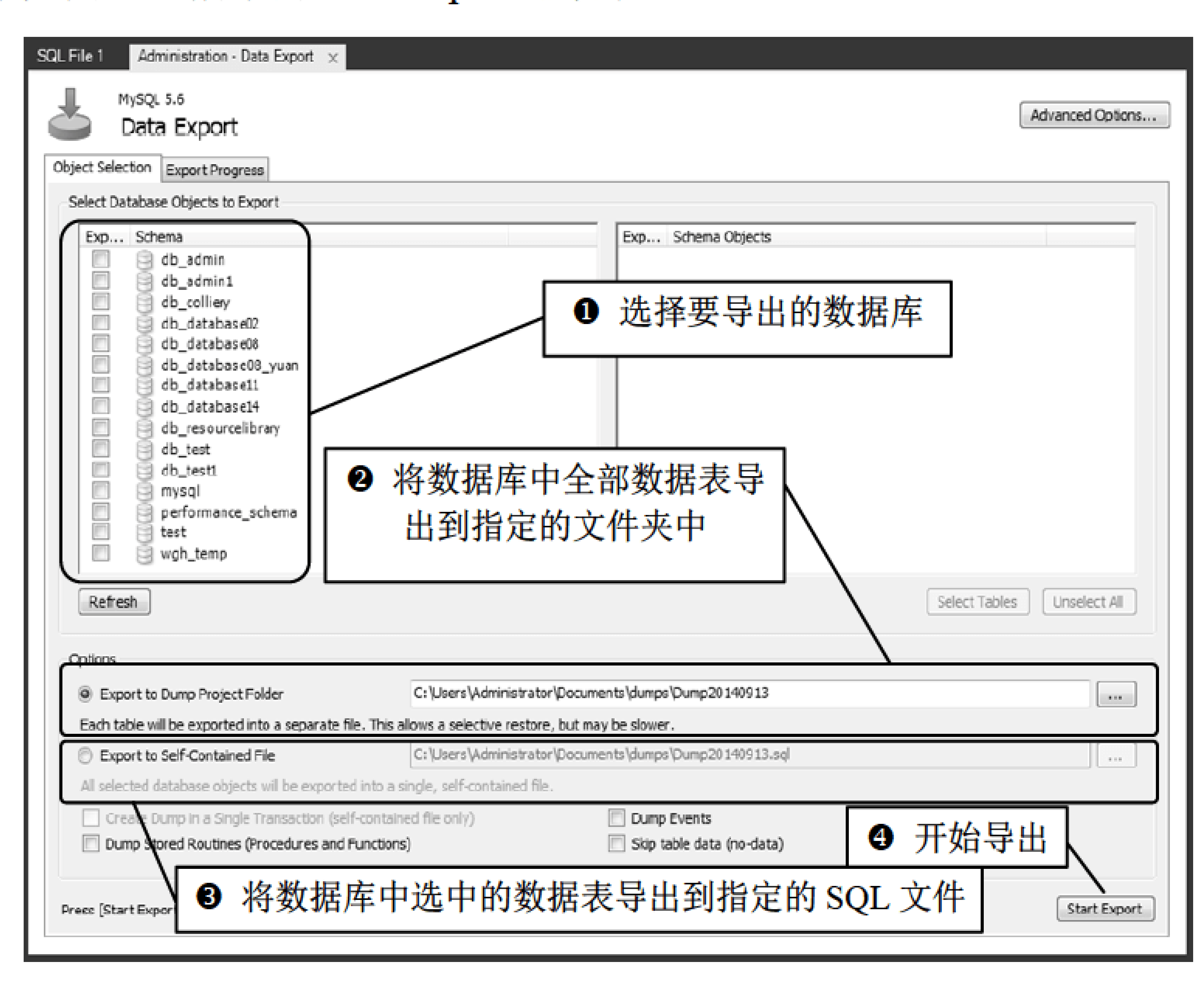


图 3.21　导出数据选项卡

（2）在导出数据选项卡中，选择要导出的数据库，如 db_database02；在下面的 Options 区域中，选中 Export to Self-Contained File 单选按钮，并且单击其右侧的“...”按钮，设置导出数据的存储位置，如图 3.22 所示。



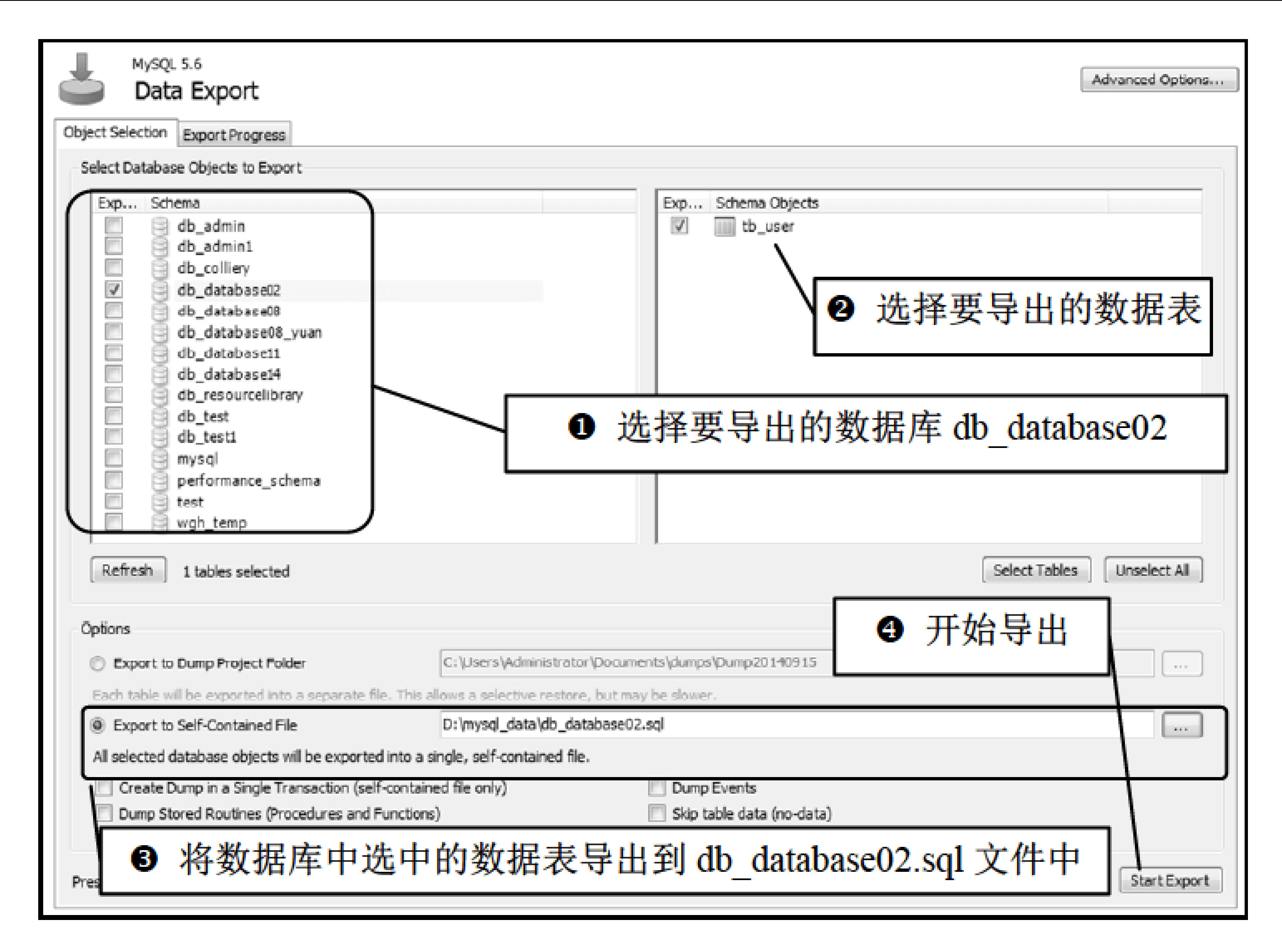


图 3.22　设置导出数据

（3）单击 Start Export 按钮，执行导出操作，导出成功后，将显示如图 3.23 所示的界面。



图 3.23　导出成功

2．数据导入

将数据库中数据导出为 SQL 文件后，还可以把它导入到 MySQL，具体步骤如下。

（1）在 MySQL Workbench 的 Local instance MySQL56 选项卡的左侧的 MANAGEMENT 栏目中，单击 Data Import/Restore 超链接，将弹出输入 root 用户密码的对话框，在该对话框中输入密码“root”，单击 OK 按钮，即可显示如图 3.24 所示的 Data Import/Restore 选项卡。



图 3.24　导入数据选项卡

（2）在导入数据选项卡中，选择根据指定的 SQL 文件导入数据。例如，从名称为 db_database02.sql 的 SQL 文件中导入数据。在 Options 区域中，选中 Import from Self-Contained File 单选按钮，并且单击其右侧的“...”按钮，选择导入 SQL 文件的存储位置，如图 3.25 所示。

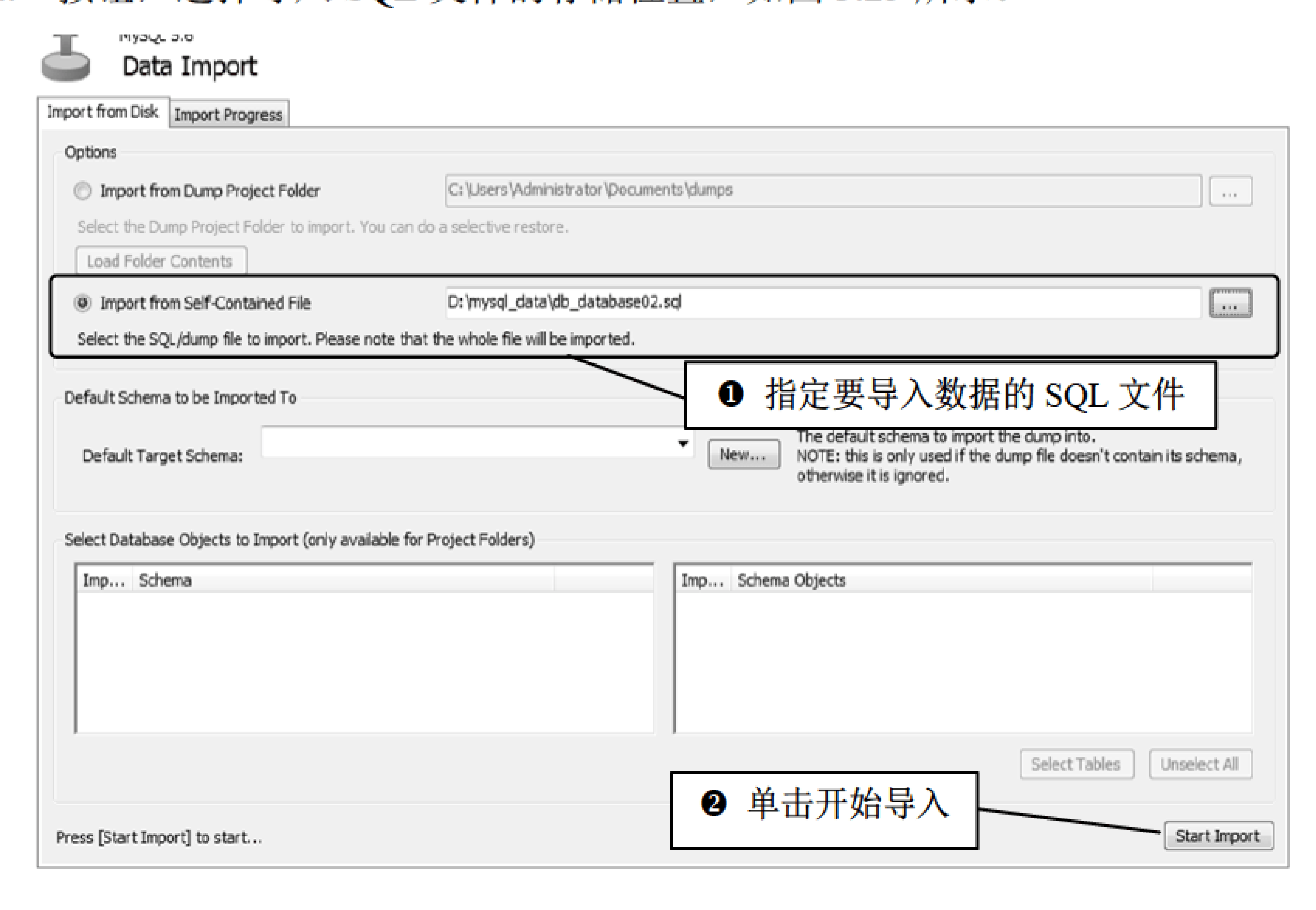
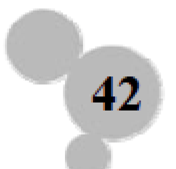


图 3.25　设置导入数据选项

（3）单击 Start Import 按钮，执行导入操作。



3.2　phpMyAdmin 图形化管理工具

phpMyAdmin 是众多 MySQL 图形化管理工具中应用最广泛的一种，是一款使用 PHP 开发的 B/S 模式的 MySQL 客户端软件，该工具是基于 Web 跨平台的管理程序，并且支持简体中文。用户可以在官方网站 www.phpmyadmin.net 上免费下载到最新的版本，本书中使用最新版本 phpMyAdmin 4.2.8。phpMyAdmin 为 Web 开发人员提供了类似于 Access、SQL Server 的图形化数据库操作界面，通过该管理工具可以完全对 MySQL 进行操作，例如，创建数据库、数据表、生成 MySQL 数据库脚本文件等。

在浏览器地址栏中输入 http://localhost/phpMyAdmin/，在弹出的对话框中输入用户名和密码，进入 phpMyAdmin 图形化管理主界面，接下来就可以进行 MySQL 数据库的操作。下面将分别介绍如何创建、修改和删除数据库。

说明

应用 phpMyAdmin 图形化管理工具有一个前提条件，必须在本机中搭建 PHP 运行环境，将其作为一个项目在 PHP 开发环境中运行应用。如果通过 PHP 的集成化安装包来搭建 PHP 运行环境，那么在这个安装包中就已经包含 phpMyAdmin 图形化管理工具，环境搭建完成后即可直接使用。

3.2.1　配置 phpMyAdmin

1．解压文件到指定目录

将下载好的 phpMyAdmin 解压缩并复制到 Web 服务器的文档根目录下。例如，使用 Apache 作为 Web 服务器，Apache 的文档根目录为 F:\wamp\webpage，则将解压好的 phpMyAdmin 文件夹复制到 F:\wamp\webpage 即可，如图 3.26 所示。

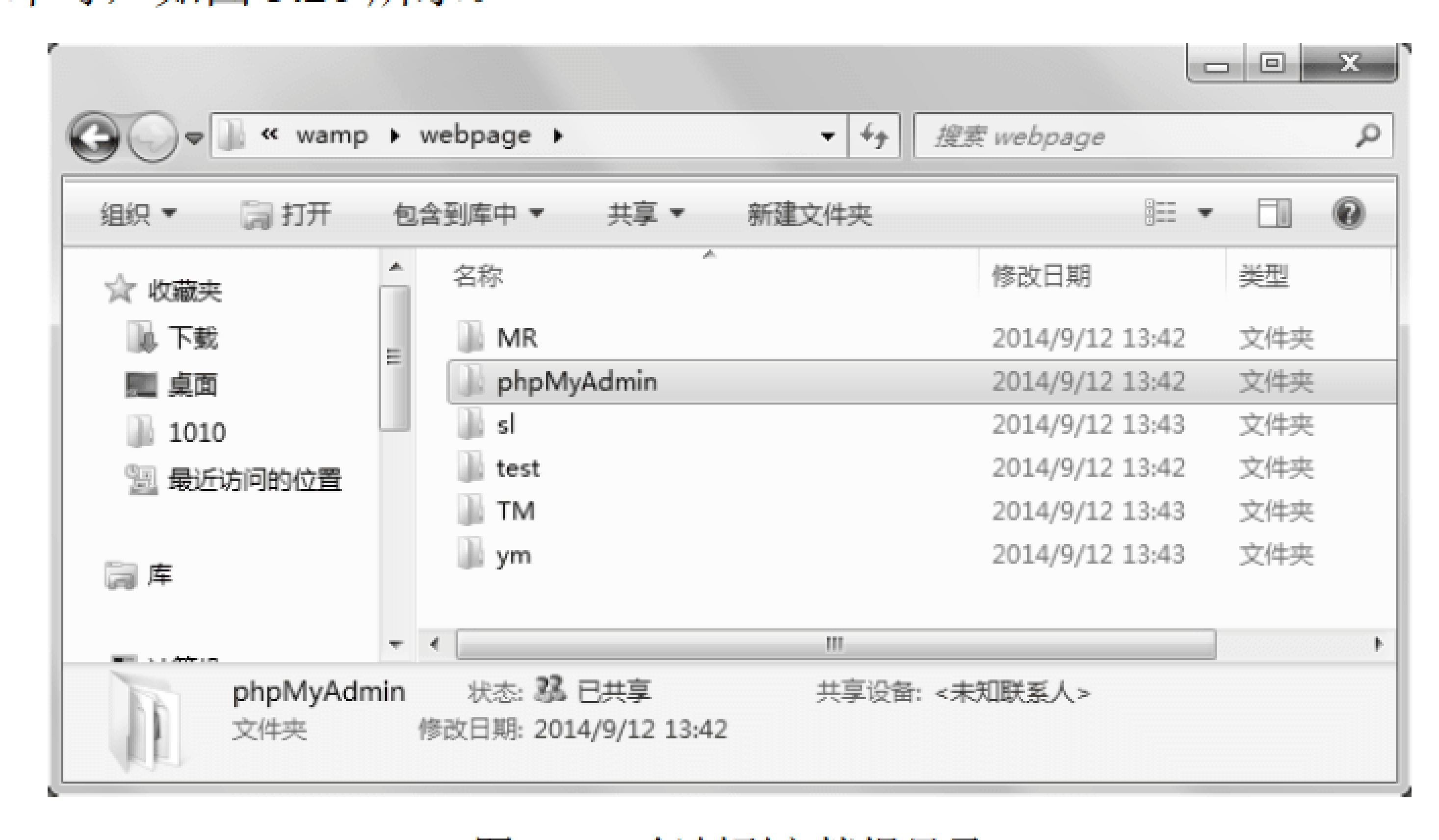


图 3.26　复制到文档根目录

2．创建 config.inc.php 文件

在 phpMyAdmin 文件夹中找到 config.sample.inc.php 文件，将其改名为 config.inc.php。

在浏览器地址栏中输入 http://localhost/phpMyAdmin，如图 3.27 所示，输入数据库的用户名密码登录。



图 3.27　登录 phpMyAdmin

登录成功可看到如图 3.28 所示界面。



图 3.28　成功登录 phpMyAdmin 界面

登录后看到有如图 3.29 所示的字样“phpMyAdmin 高级功能尚未完全设置，部分功能未激活，请点击这里查看原因”，则执行如下步骤。

> ⚠ phpMyAdmin 高级功能尚未完全设置，部分功能未激活。请点击这里查看原因。

图 3.29　phpMyadmin 高级功能尚未完全设置提示

（1）单击“导入”标签，然后单击“浏览”按钮，找到 phpMyAdmin/examples/create_table.sql 文件，将它导入，如图 3.30 和图 3.31 所示。



图 3.30　导入 create_table.sql 文件

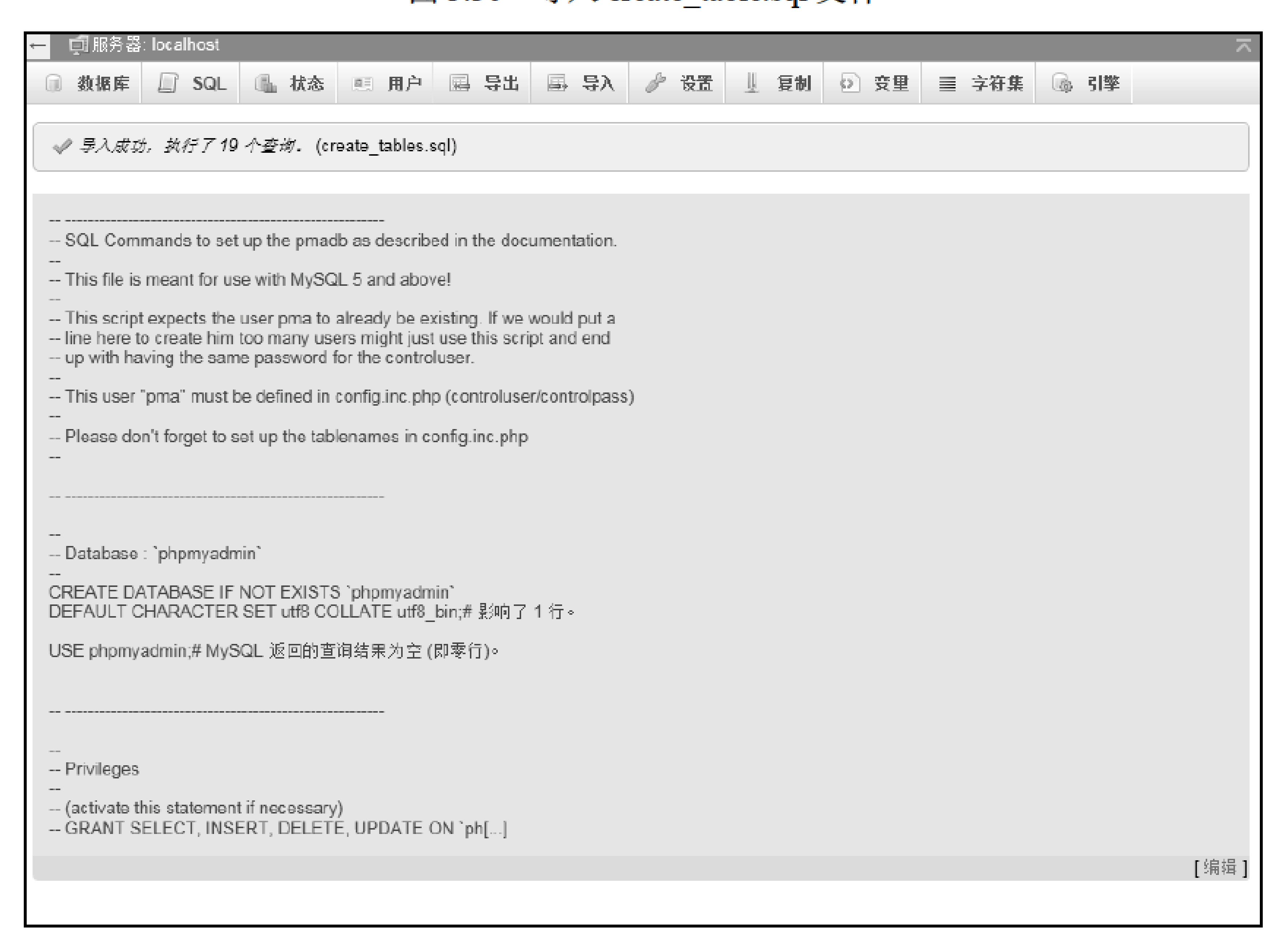


图 3.31　导入 create_table.sql 文件成功

（2）将 config.inc.php 文件中的以下内容的注释去掉。

```
$cfg['Servers'][$i]['pmadb'] = 'phpmyadmin';
$cfg['Servers'][$i]['bookmarktable'] = 'pma__bookmark';
$cfg['Servers'][$i]['relation'] = 'pma__relation';
$cfg['Servers'][$i]['table_info'] = 'pma__table_info';
$cfg['Servers'][$i]['table_coords'] = 'pma__table_coords';
$cfg['Servers'][$i]['pdf_pages'] = 'pma__pdf_pages';
$cfg['Servers'][$i]['column_info'] = 'pma__column_info';
$cfg['Servers'][$i]['history'] = 'pma__history';
$cfg['Servers'][$i]['table_uiprefs'] = 'pma__table_uiprefs';
$cfg['Servers'][$i]['tracking'] = 'pma__tracking';
$cfg['Servers'][$i]['designer_coords'] = 'pma__designer_coords';
$cfg['Servers'][$i]['userconfig'] = 'pma__userconfig';
$cfg['Servers'][$i]['recent'] = 'pma__recent';
$cfg['Servers'][$i]['favorite'] = 'pma__favorite';
$cfg['Servers'][$i]['users'] = 'pma__users';
$cfg['Servers'][$i]['usergroups'] = 'pma__usergroups';
$cfg['Servers'][$i]['navigationhiding'] = 'pma__navigationhiding';
$cfg['Servers'][$i]['savedsearches'] = 'pma__savedsearches';
```

（3）修改 phpMyAdmin/libraries/config.default.php 文件，内容如下。

```
$cfg['Servers'][$i]['controluser'] = 'username';      /*数据库用户名*/
$cfg['Servers'][$i]['controlpass'] = 'password';      /*数据库密码*/
$cfg['Servers'][$i]['pmadb'] = 'phpmyadmin';
$cfg['Servers'][$i]['bookmarktable'] = 'pma_bookmark';
$cfg['Servers'][$i]['relation'] = 'pma_relation';
$cfg['Servers'][$i]['table_info'] = 'pma_table_info';
$cfg['Servers'][$i]['table_coords'] = 'pma_table_coords';
$cfg['Servers'][$i]['pdf_pages'] = 'pma_pdf_pages';
$cfg['Servers'][$i]['column_info'] = 'pma_column_info';
$cfg['Servers'][$i]['history'] = 'pma_history';
$cfg['Servers'][$i]['designer_coords'] = 'pma_designer_coords';
$cfg['Servers'][$i]['recent'] = 'pma_recent';
$cfg['Servers'][$i]['table_uiprefs'] = 'pma_table_uiprefs';
$cfg['Servers'][$i]['tracking'] = 'pma_tracking';
$cfg['Servers'][$i]['userconfig'] = 'pma_userconfig';
```

3.2.2 数据库操作管理

1．创建数据库

在 phpMyAdmin 的主界面，首先在文本框中输入数据库的名称 db_study，然后在下拉列表框中选择所要使用的编码，一般选择 gb2312_chinese_ci 简体中文编码，单击“创建”按钮，创建数据库，如图 3.32 所示。成功创建数据库后，将显示如图 3.33 所示的界面。



图 3.32　创建数据库

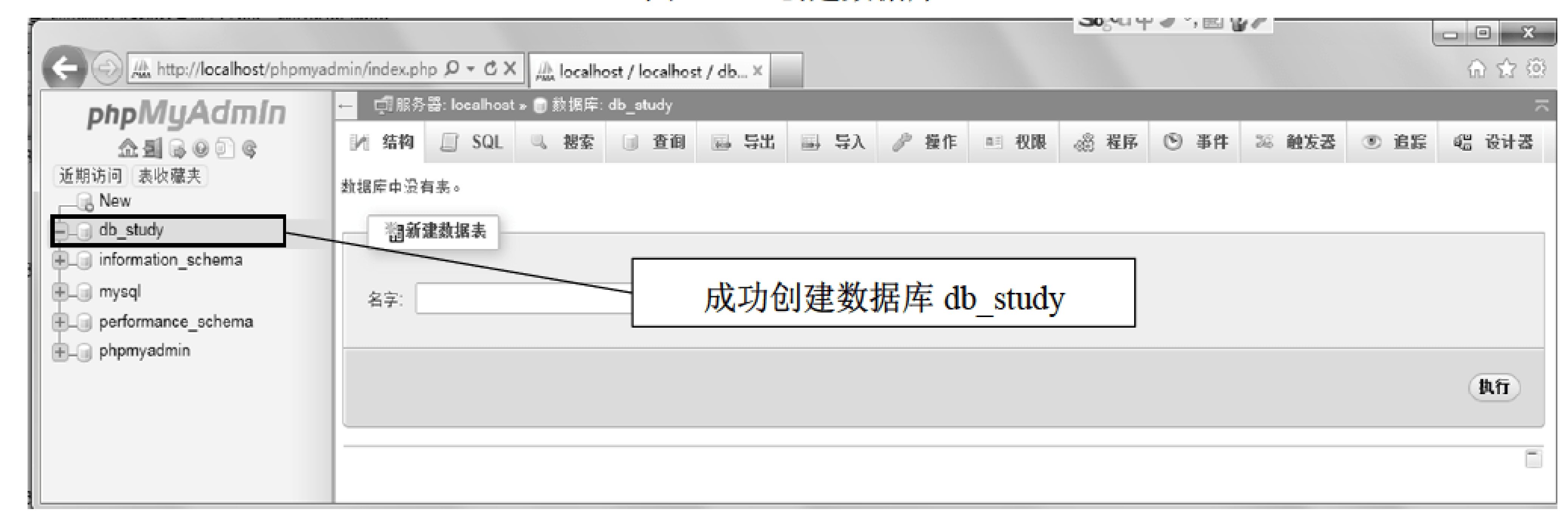


图 3.33　数据库创建成功

说明

在右侧界面中可以对该数据库进行相关操作，如结构、SQL、搜索、查询、导出、导入、权限等，单击相应的标签进入相应的操作界面。

2．修改、删除数据库

在如图 3.34 所示的界面中，在右侧界面还可以对当前数据库进行修改。单击界面中的“操作”标签，进入修改操作页面。

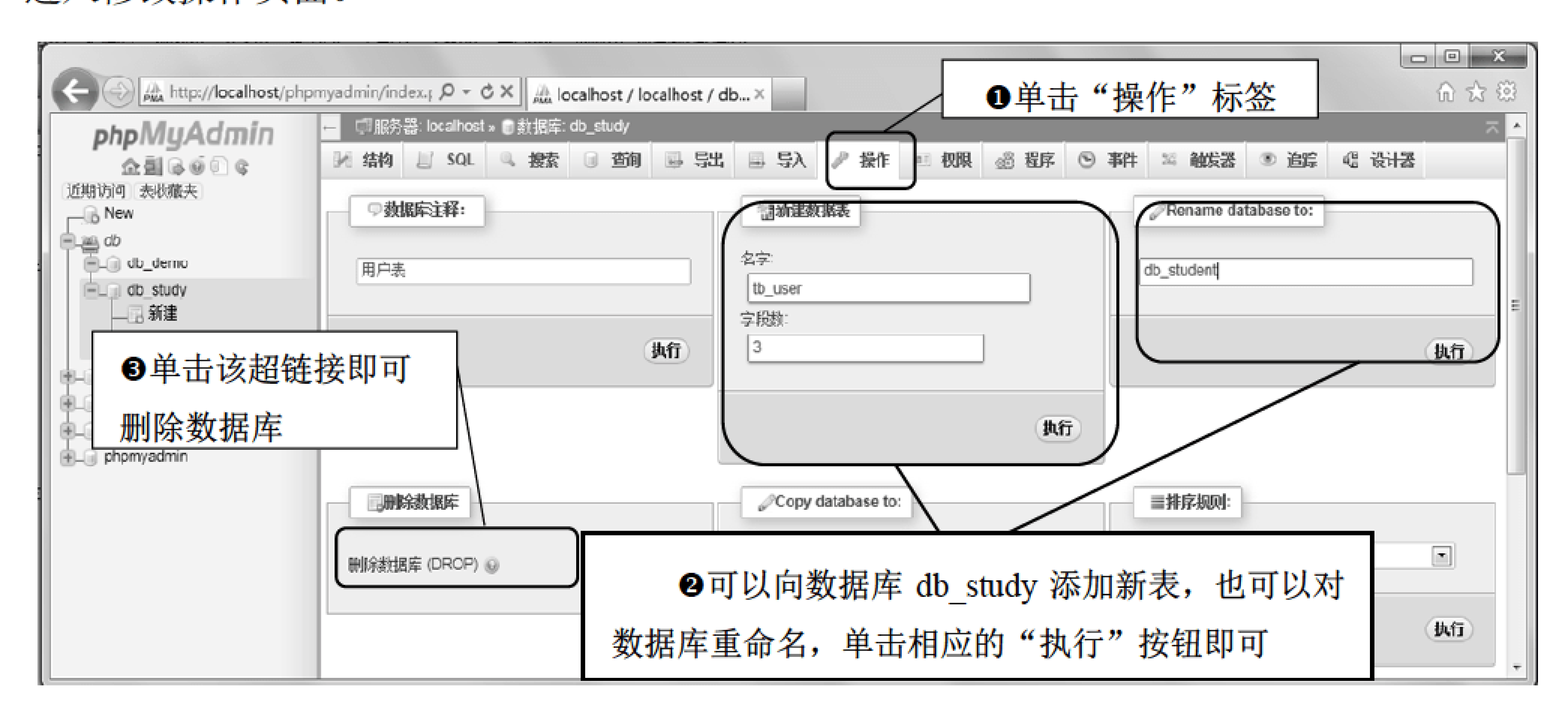


图 3.34　修改数据库

（1）可以对当前数据库执行创建数据表的操作，只要在创建数据表的提示信息下面的两个文本框中分别输入要创建的数据表的名称和字段总数，然后单击“执行”按钮即可进入到创建数据表结构页面。

（2）也可以对当前的数据库重命名，在 Rename database to:下的文本框中输入新的数据库名称，单击“执行”按钮，即可成功修改数据库名称。

（3）也可以单击“删除数据库”超链接来删除该数据库。

3.2.3　管理数据表

管理数据表是以选择指定的数据库为前题，然后在该数据库中创建并管理数据表。下面就来介绍如何创建、修改和删除数据表。

1．创建数据表

创建数据库 db_study 后，在右侧的操作页面中输入数据表的名称和字段数，然后单击“执行”按钮，即可创建数据表，如图 3.35 所示。

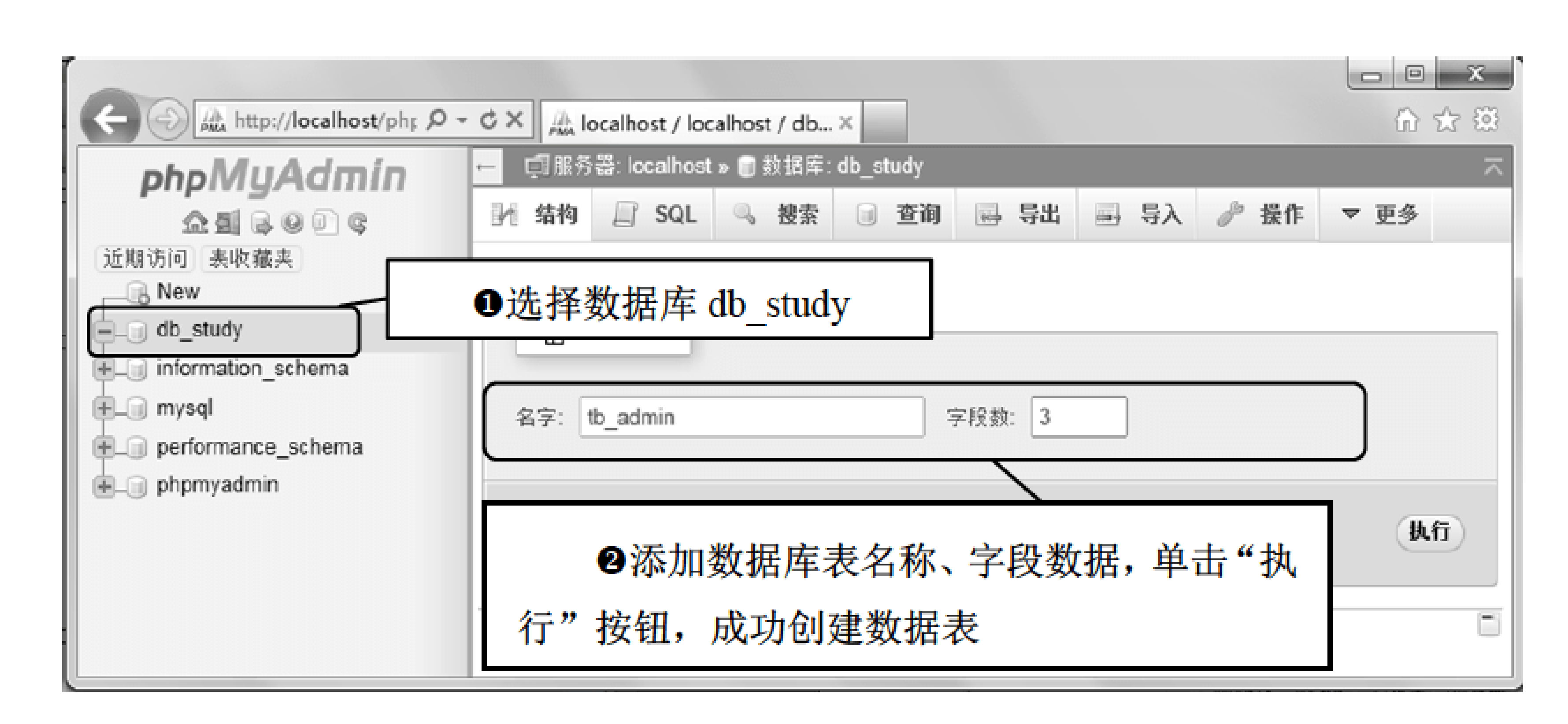


图 3.35　创建数据表

成功创建数据表 tb_admin 后，将显示数据表结构界面。在表单中对各个字段的详细信息进行录入，包括字段名、数据类型、长度/值、是否为空、主键、是否自增等，以完成对表结构的详细设置。当所有的信息都输入以后，单击“保存”按钮，创建数据表结构，如图 3.36 所示。成功创建数据表结构后，将显示如图 3.37 所示的界面。

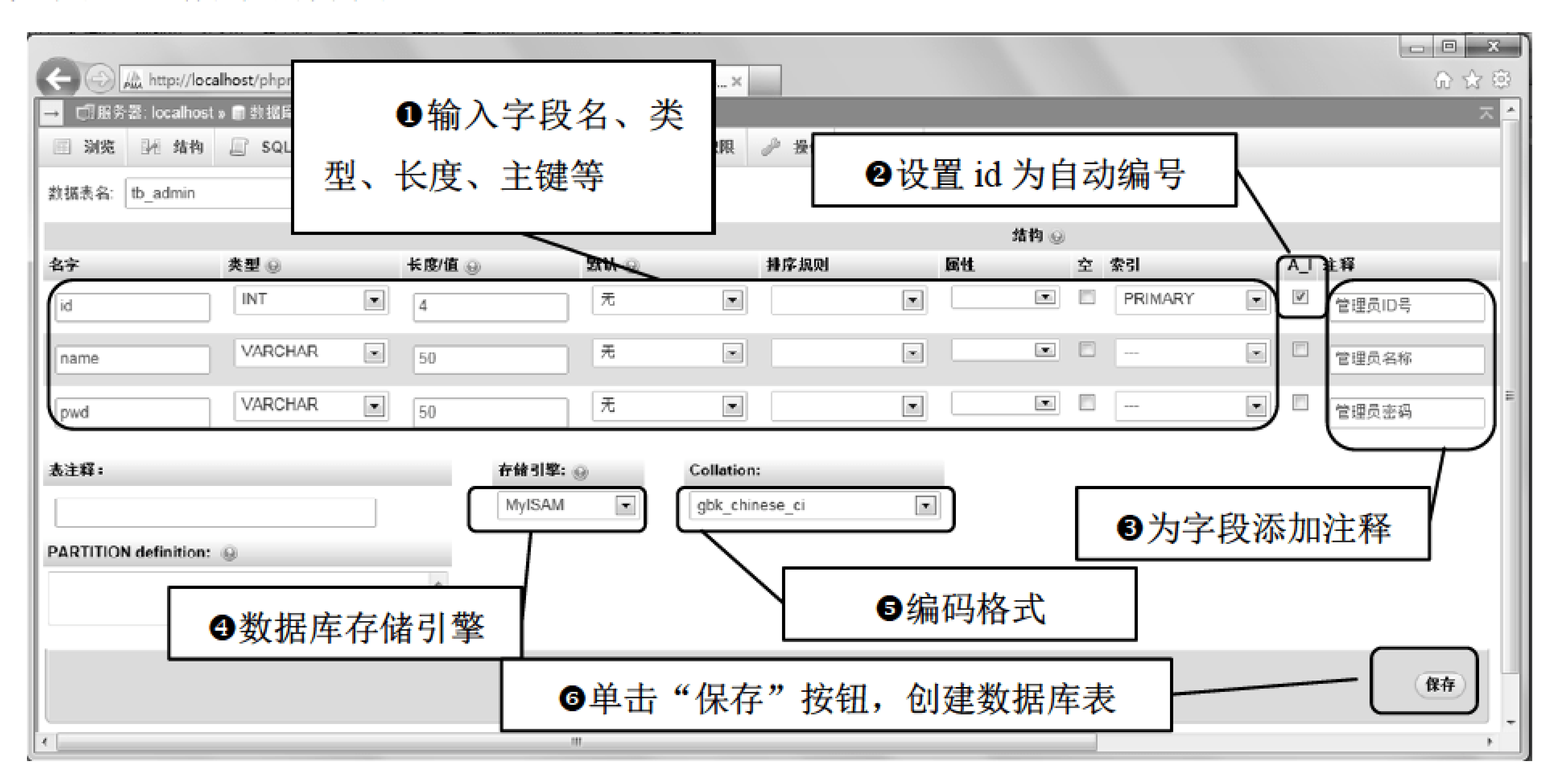


图 3.36　创建数据表结构



图 3.37　成功创建数据表

2. 修改数据表

一个新的数据表被创建后，进入到数据表页面中，在这里可以通过改变表的结构来修改表，可以执行添加新的列、删除列、索引列、修改列的数据类型或者字段的长度/值等操作，如图 3.38 所示。

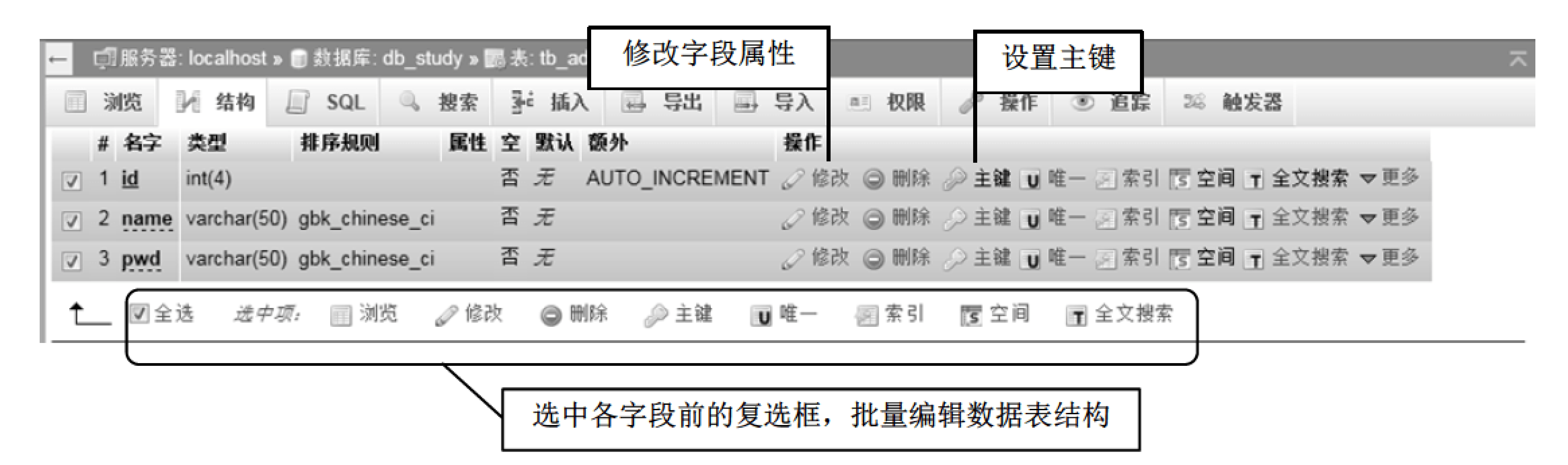


图 3.38　修改数据表结构

3. 删除数据表

要删除某个数据表，单击页面中的“操作”标签，之后单击页面中红色的超链接“删除数据表”即可成功删除指定的数据表，如图 3.39 所示。



图 3.39　删除数据表结构

3.2.4　管理数据记录

单击 phpMyAdmin 主界面中的 SQL 标签，打开 SQL 语句编辑区。在编辑区输入完整的 SQL 语句，



来实现数据的查询、插入、修改和删除等操作。

1．使用 SQL 语句插入数据

在 SQL 语句编辑区使用 INSERT 语句向数据表 tb_admin 中插入数据后，单击“执行”按钮，向数据表中插入一条数据，如图 3.40 所示。如果提交的 SQL 语句有错误，系统会给出一个警告，提示用户修改它；如果提交的 SQL 语句正确，则弹出如图 3.41 所示的提示信息。



图 3.40　使用 SQL 语句向数据表中插入数据



图 3.41　成功添加数据信息



为了编写方便，可以利用其右侧的属性列表来选择要操作的列，只要选中要添加的列，双击其选项或者单击<<按钮添加列名称。

2．使用 SQL 语句修改数据

在 SQL 语句编辑区应用 UPDATE 语句修改数据信息，将 ID 为 1 的管理员的名称改为“明日科技”，密码改为 111，添加的 SQL 语句如图 3.42 所示。



图 3.42　修改数据信息的 SQL 语句

单击“执行”按钮，数据修改成功。比较修改前后的数据如图 3.43 所示。

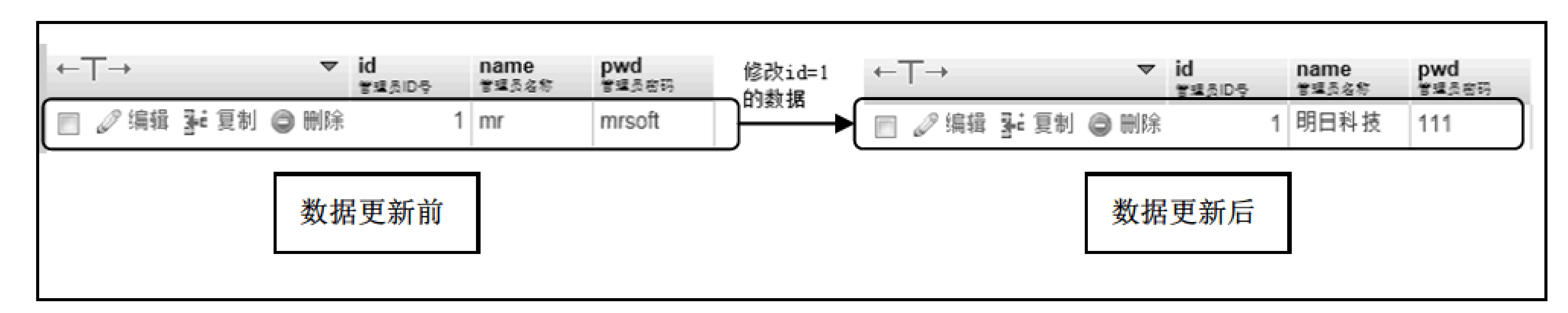


图 3.43　修改单条数据的实现过程

3．使用 SQL 语句查询数据

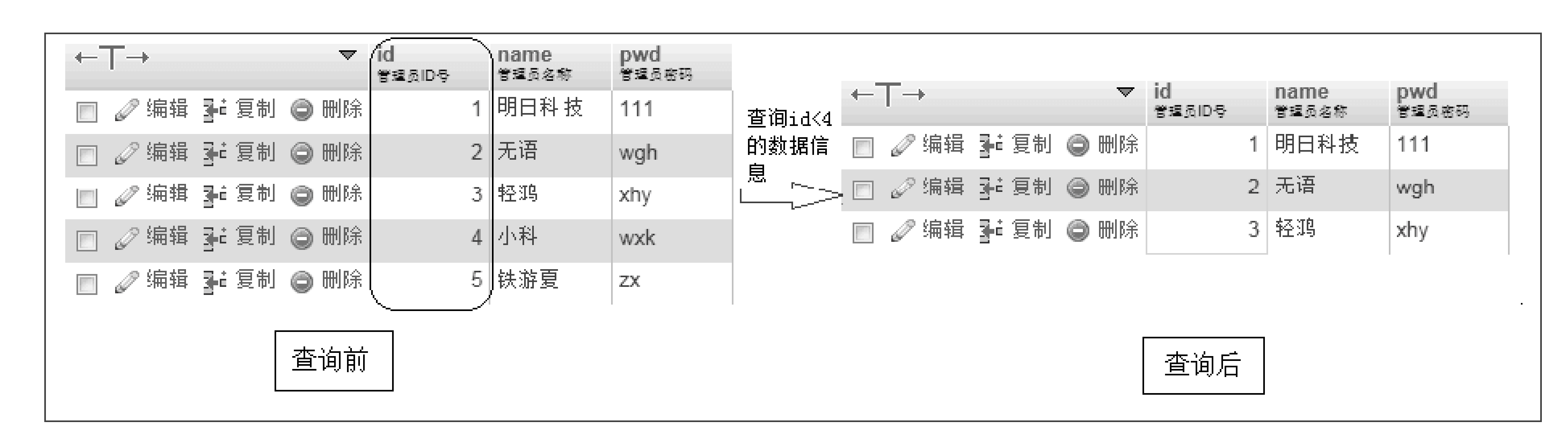
在 SQL 语句编辑区应用 SELECT 语句检索指定条件的数据信息，将 ID 小于 4 的管理员全部显示出来，添加的 SQL 语句如图 3.44 所示。



图 3.44　查询数据信息的 SQL 语句



单击“执行”按钮，该语句的实现过程如图 3.45 所示。



id 管理员ID号	name 管理员名称	pwd 管理员密码
1	明日科技	111
2	无语	wgh
3	轻鸿	xhy
4	小科	wxk
5	铁游夏	zx

id 管理员ID号	name 管理员名称	pwd 管理员密码
1	明日科技	111
2	无语	wgh
3	轻鸿	xhy

图 3.45　查询指定条件的数据信息的实现过程

除了对整个表的简单查询外，还可以执行复杂的条件查询（使用 WHERE 子句提交 LIKE、ORDER BY、GROUP BY 等条件查询语句）及多表查询，读者可通过实践灵活运用 SQL 语句功能。

4．使用 SQL 语句删除数据

在 SQL 语句编辑区应用 delete 语句检索指定条件的数据或全部数据信息，删除名称为“小科”的管理员信息，添加的 SQL 语句如图 3.46 所示。



图 3.46　删除指定数据信息的 SQL 语句

注意

如果 DELETE 语句后面没有 WHERE 条件值，那么将删除指定数据表中的全部数据。

单击“执行”按钮，弹出确认删除操作对话框，单击“确定”按钮，执行数据表中指定条件的删除操作。该语句的实现过程如图 3.47 所示。

管理员信息表数据

id 管理员ID号	name 管理员名称	pwd 管理员密码
2	轻鸿	xhy
3	无语	wgh
4	小科	wxk
5	铁游夏	zx

删除 name='小科'的数据信息

删除数据后的结果

id 管理员ID号	name 管理员名称	pwd 管理员密码
2	轻鸿	xhy
3	无语	wgh
5	铁游夏	zx

图 3.47　删除指定条件的数据信息的实现过程

5. 通过 form 表单插入数据

选择某个数据表后，单击“插入”标签，进入插入数据界面，如图 3.48 所示。在界面中输入各字段值，单击“执行”按钮即可插入记录。默认情况下，一次可以插入两条记录。

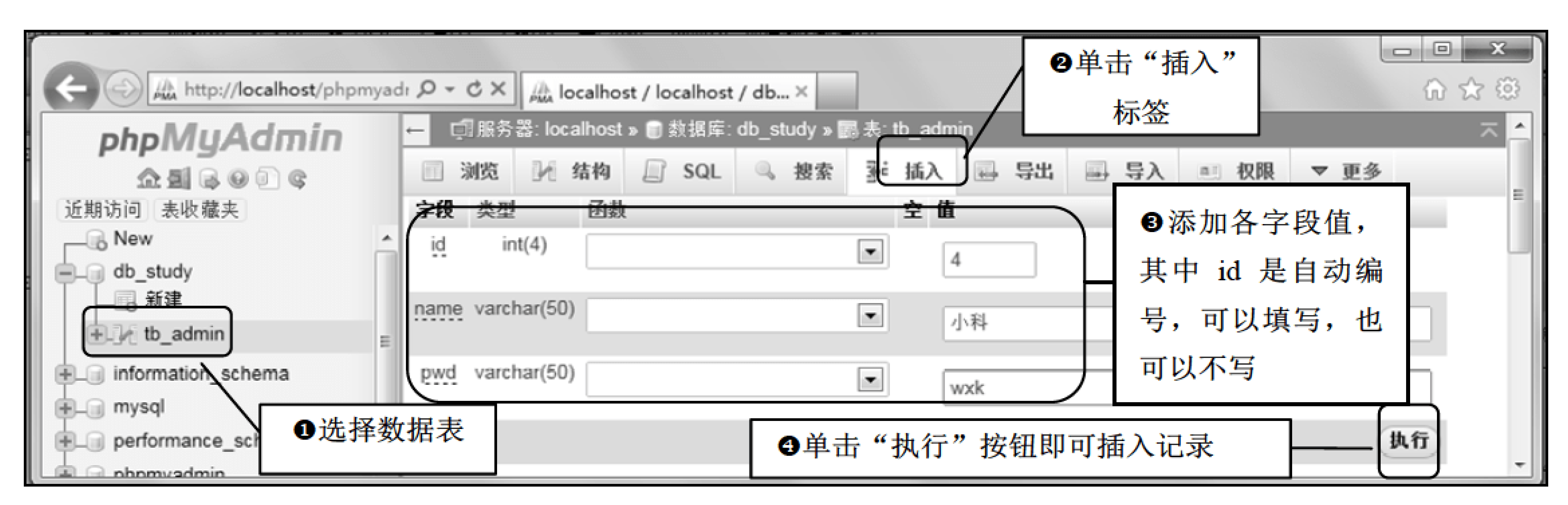


图 3.48　插入数据

6. 浏览数据

选择某个数据表后，单击“浏览”标签，进入浏览界面，如图 3.49 所示。单击每行记录中的编辑超链接，可以对该记录进行编辑；单击每行记录中的删除超链接，可以删除该条记录。



图 3.49　浏览数据

7. 搜索数据

选择某个数据表后，单击“搜索”标签超级链接，进入搜索页面，如图 3.50 所示。在这个页面中，



可以在选择字段的列表框中选择一个或多个列，如果要选择多个列，先按下 Ctrl 键并单击要选择的字段名，查询结果将按照选择的字段名进行输出。

在该界面中可以对记录按条件进行查询。查询方式有两种：第一种方式使用依例查询。选择要查询的条件，并在文本框中输入要查询的值，单击“执行”按钮。第二种方式选择构建 WHERE 语句查询。直接在“where 语句的主体”文本框中输入查询语句，然后单击其后的“执行”按钮。

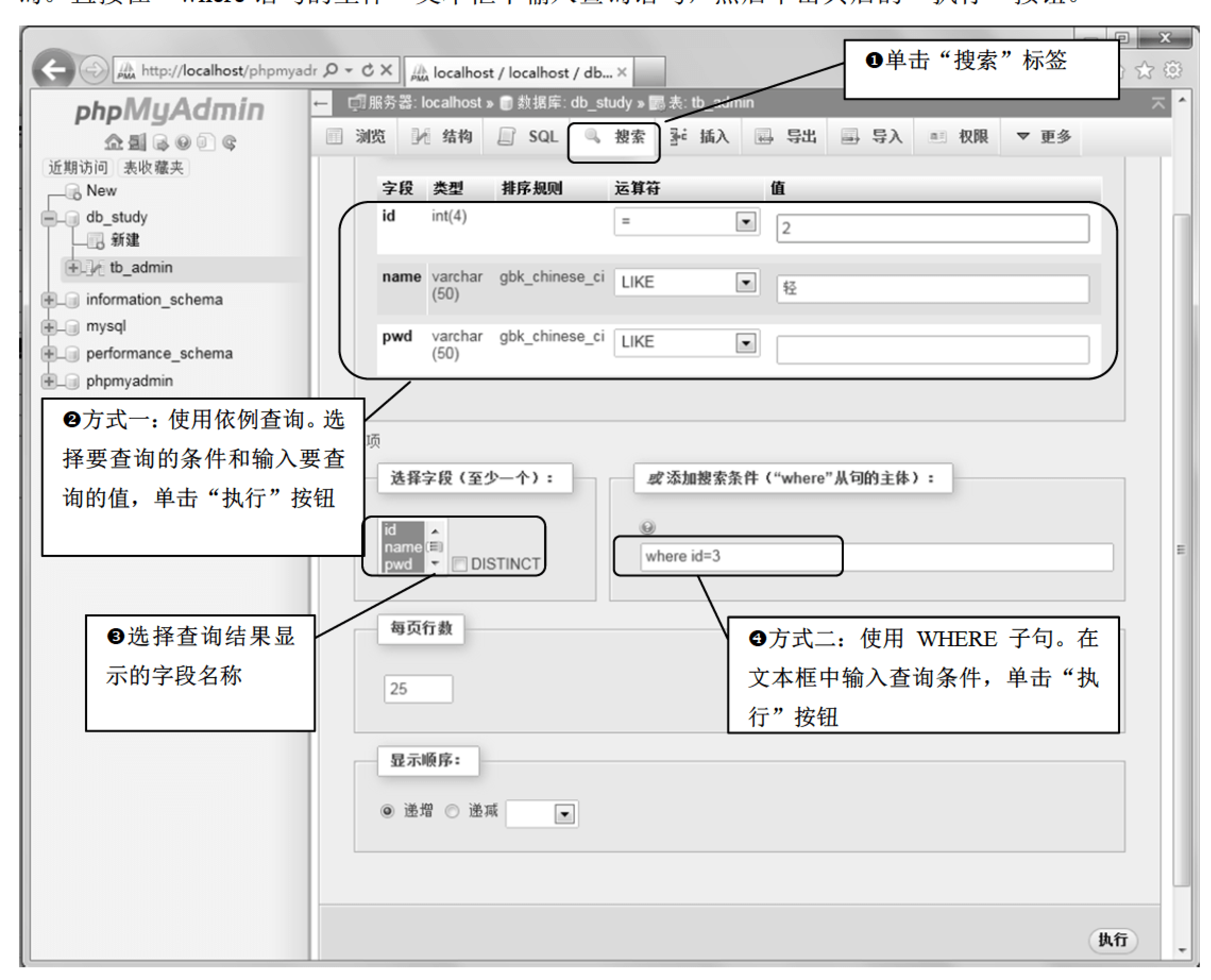


图 3.50　搜索查询

3.2.5　导出导入数据

导出和导入 MySQL 数据库脚本是互逆的两个操作。导出是将数据表结构、表记录存储为.sql 的脚本文件；导入是执行扩展名为“.sql”的文件，将数据导入到数据库中。通过导入和导出的操作实现数据库的备份和还原。

1. 导出 MySQL 数据库脚本

单击 phpMyAdmin 主界面中的“导出”标签，打开导出编辑区，如图 3.51 所示。选择导出文件的格式，这里默认使用选项 SQL，单击“执行”按钮，弹出如图 3.52 所示的“另存为”对话框，单击“保存”按钮，将脚本文件以“.sql”格式存储在指定位置。

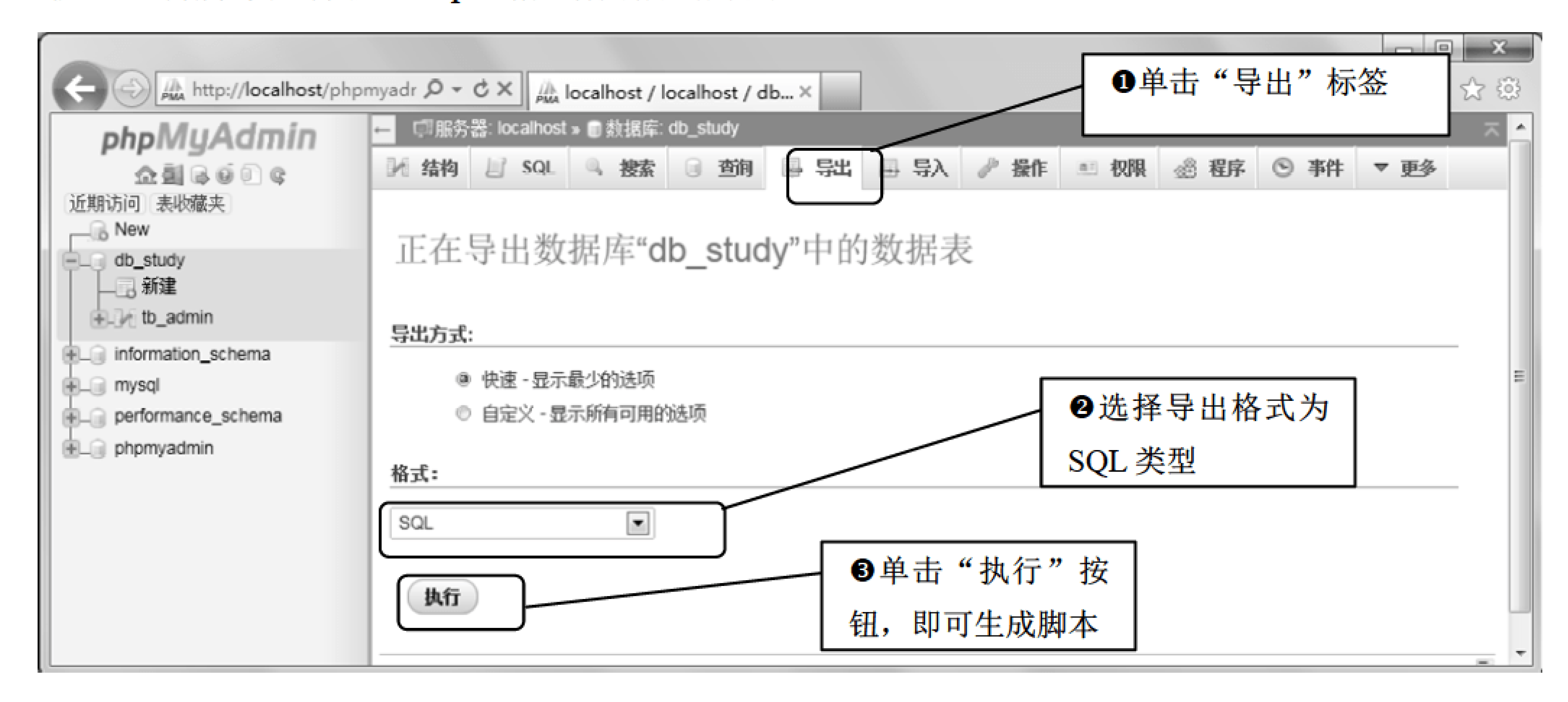


图 3.51　生成 MySQL 脚本文件设置界面

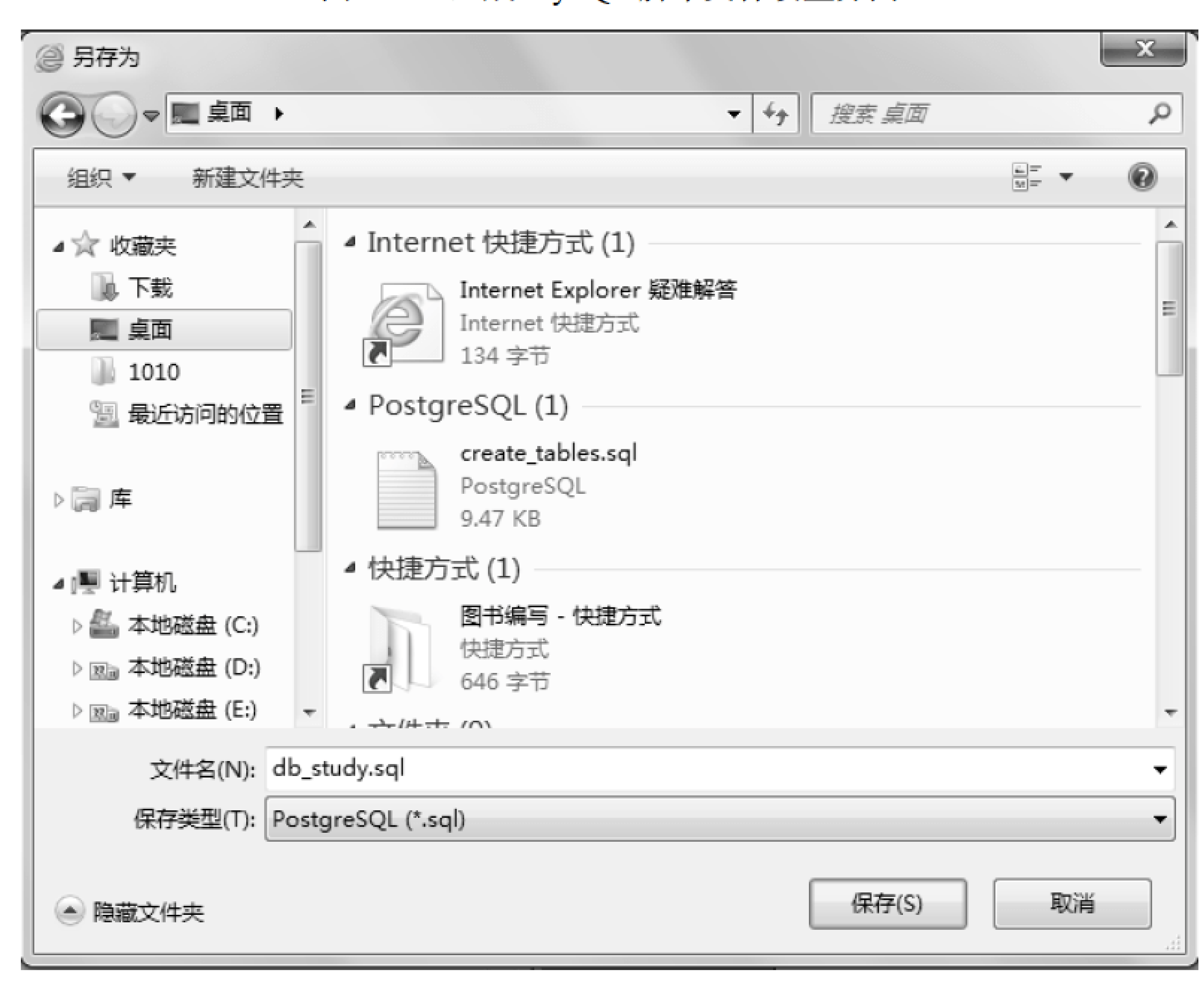


图 3.52　另存为 MySQL 脚本对话框

2. 导入 MySQL 数据库脚本

单击“导入”标签，进入执行 MySQL 数据库脚本界面，单击“浏览”按钮查找脚本文件（如 db_study.sql）所在位置，如图 3.53 所示，单击“执行”按钮，即可执行 MySQL 数据库脚本文件。

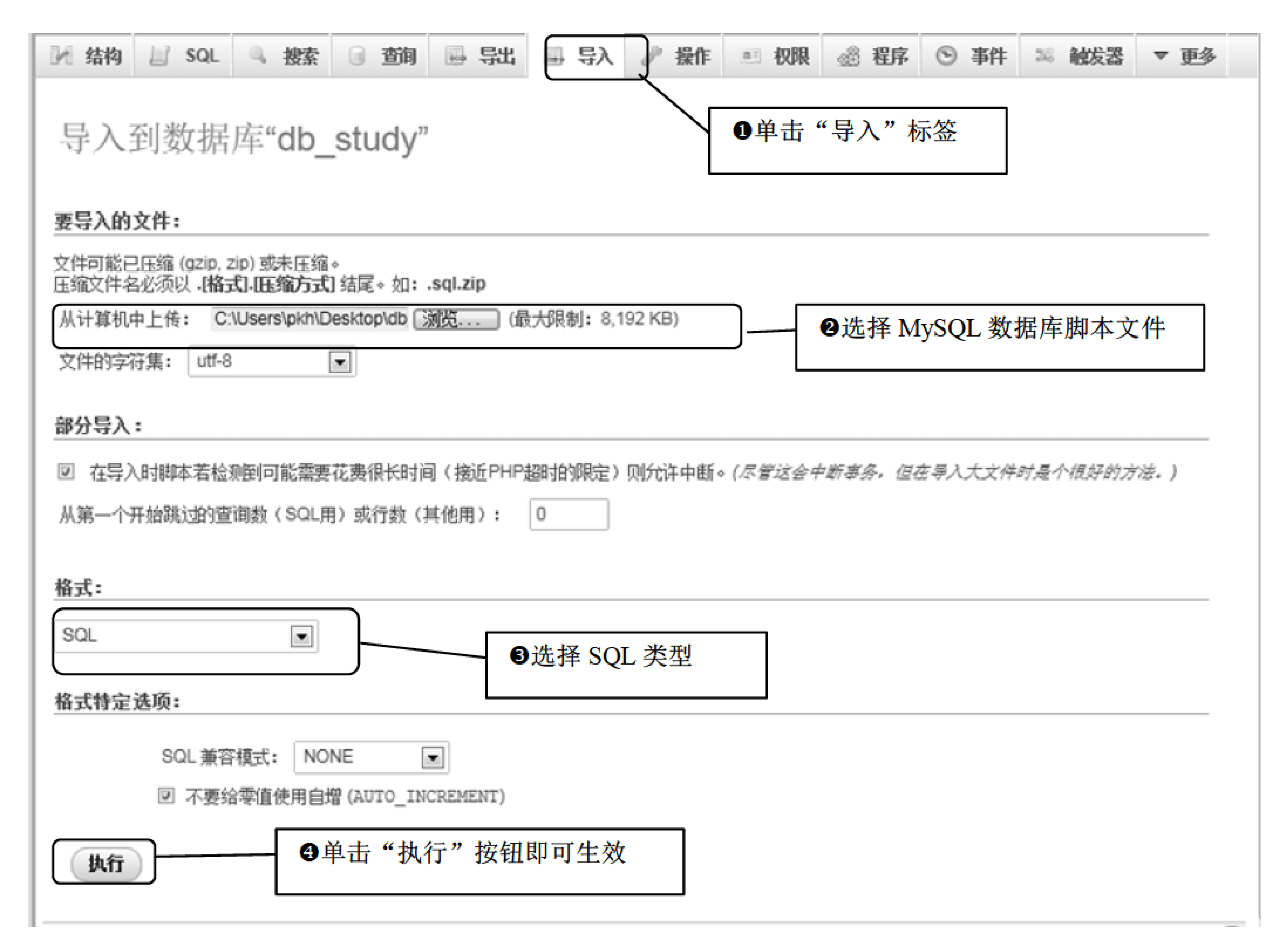


图 3.53 执行 MySQL 数据库脚本文件

注意

在执行 MySQL 脚本文件前，首先检测是否有与所导入数据库同名的数据库，如果没有同名的数据库，则首先要在数据库中创建一个名称与数据文件中的数据库名相同的数据库，然后再执行 MySQL 数据库脚本文件。另外，在当前数据库中，不能有与将要导入数据库中的数据表重名的数据表存在，如果有重名的表存在导入文件就会失败，提示错误信息。

说明

读者可也可通过单击 phpMyAdmin 图形化工具左侧区的按钮，在打开的对话框中，单击“导入文件”超链接，然后选择脚本文件所在的位置，从而执行脚本文件。

3.2.6 phpMyAdmin 设置编码格式

将页面、程序文件、数据库与数据表设置统一的编码格式可以使程序运行时不至于出现乱码。一般情况下，设置页面的编码格式由 HTML 中的 meta 标签实现，设置程序文件的编码格式是由 header() 函数实现，设置数据库与数据表的编码格式可以通过使用 phpMyAdmin 实现。下面以实例详细讲解一下如何为新创建的数据库设置编码格式。具体步骤如下。

（1）登录到 phpMyAdmin 图形化工具页面，创建数据库名称，并为新创建的数据库选择编码格式，如图 3.54 所示。

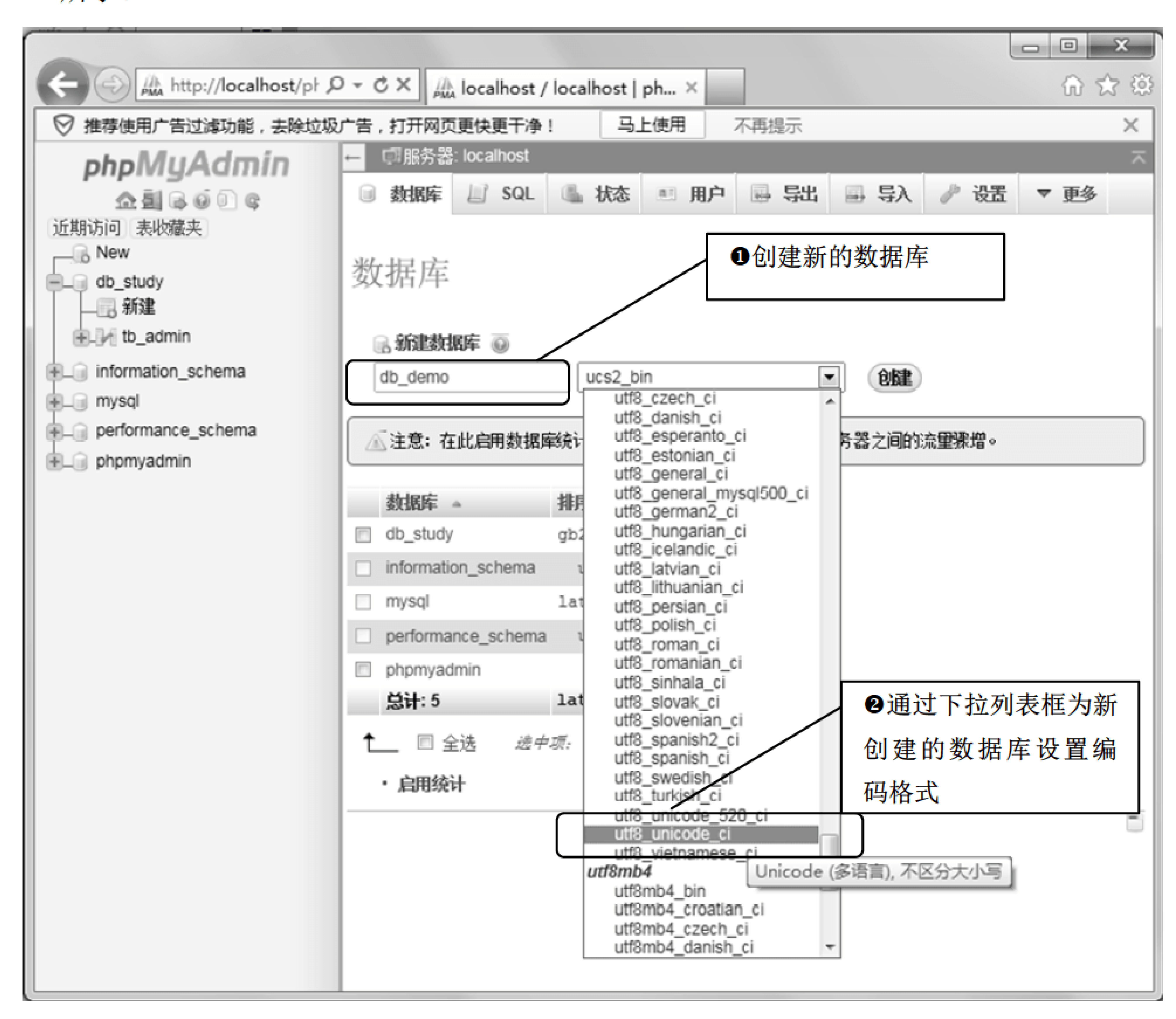


图 3.54　设置数据库的编码格式

（2）创建数据表，定义数据表字段，并为新创建的数据表设置编码格式，如图 3.55 所示。



图 3.55　设置字段编码格式

3.2.7　phpMyAdmin 添加服务器新用户

在 phpMyAdmin 图形化管理工具中，不但可以对 MySQL 数据库进行各种操作，而且可以添加服务器的新用户，并对新添加的用户设置权限。

在 phpMyAdmin 中添加 MySQL 服务器新用户的步骤如下。

（1）单击 phpMyAdmin 主界面中的“用户”标签，打开服务器用户操作界面，如图 3.56 所示。



图 3.56　服务器用户一览表

（2）在该界面中，单击“添加用户”超链接。进入到如图 3.57 所示界面，设置用户名、密码、主机，并对新用户的权限进行设置。设置完成后，单击“执行”按钮，完成对新用户的添加操作，返回主页面，将提示新用户添加成功。

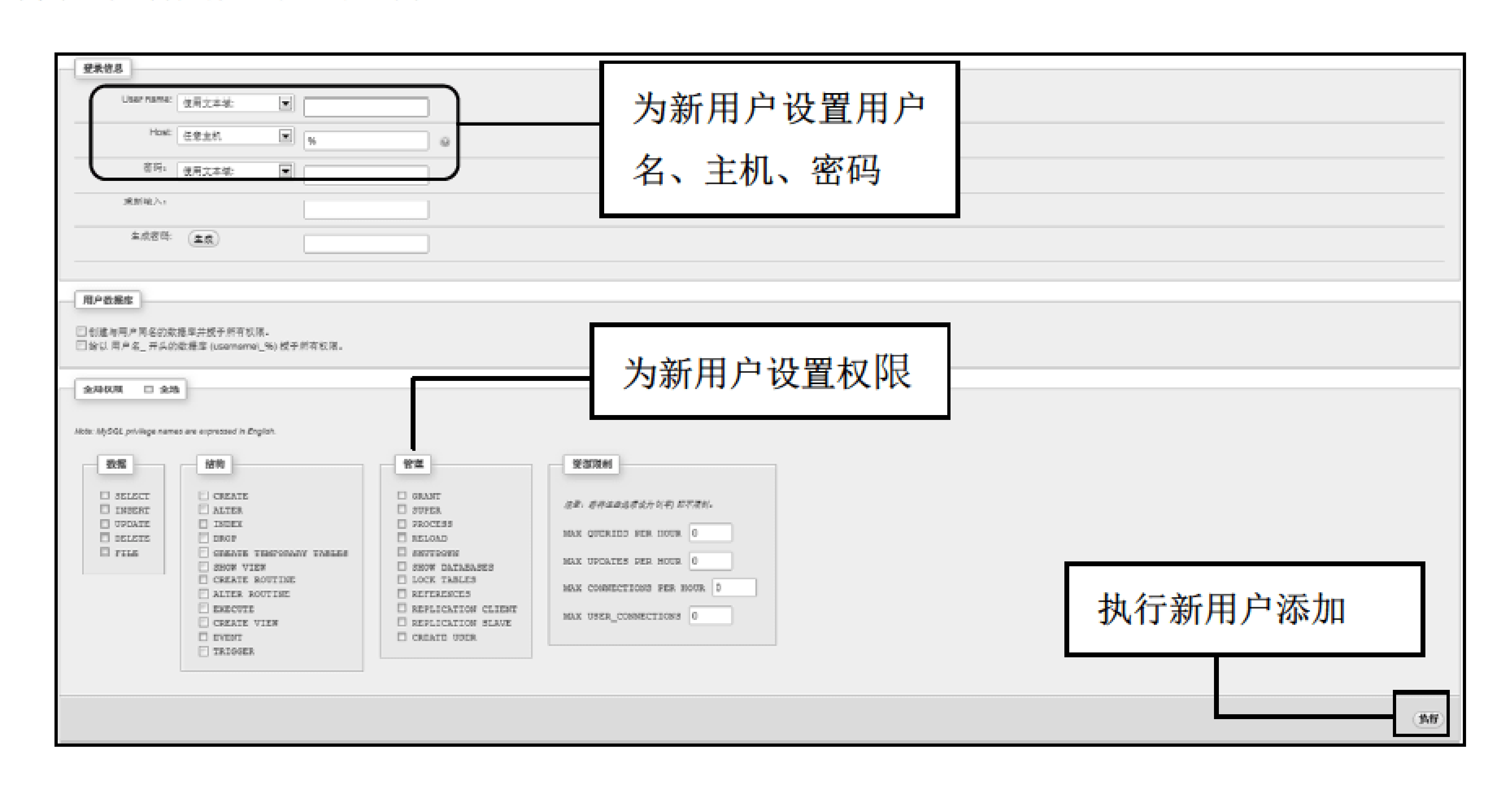


图 3.57　设置添加用户信息

3.2.8　phpMyAdmin 中重置 MySQL 服务器登录密码

在 phpMyAdmin 图形化管理工具中，不但可以对 MySQL 数据库进行各种操作，而且可以对用户的权限进行设置，同时还可以对 MySQL 服务器的登录密码进行重置。

在 phpMyAdmin 中重置 MySQL 服务器登录密码的步骤如下。

（1）单击服务器用户操作界面，如图 3.58 所示。



图 3.58　服务器用户一览表



（2）在该界面中，可以对指定用户的权限进行编辑、可以添加新用户和删除指定的用户。这里选择指定的用户，单击 编辑权限 超链接，对指定用户的权限进行设置，进入到如图 3.59 所示界面。



图 3.59　编辑用户权限

单击“修改密码”按钮，进入到如图 3.60 所示界面。

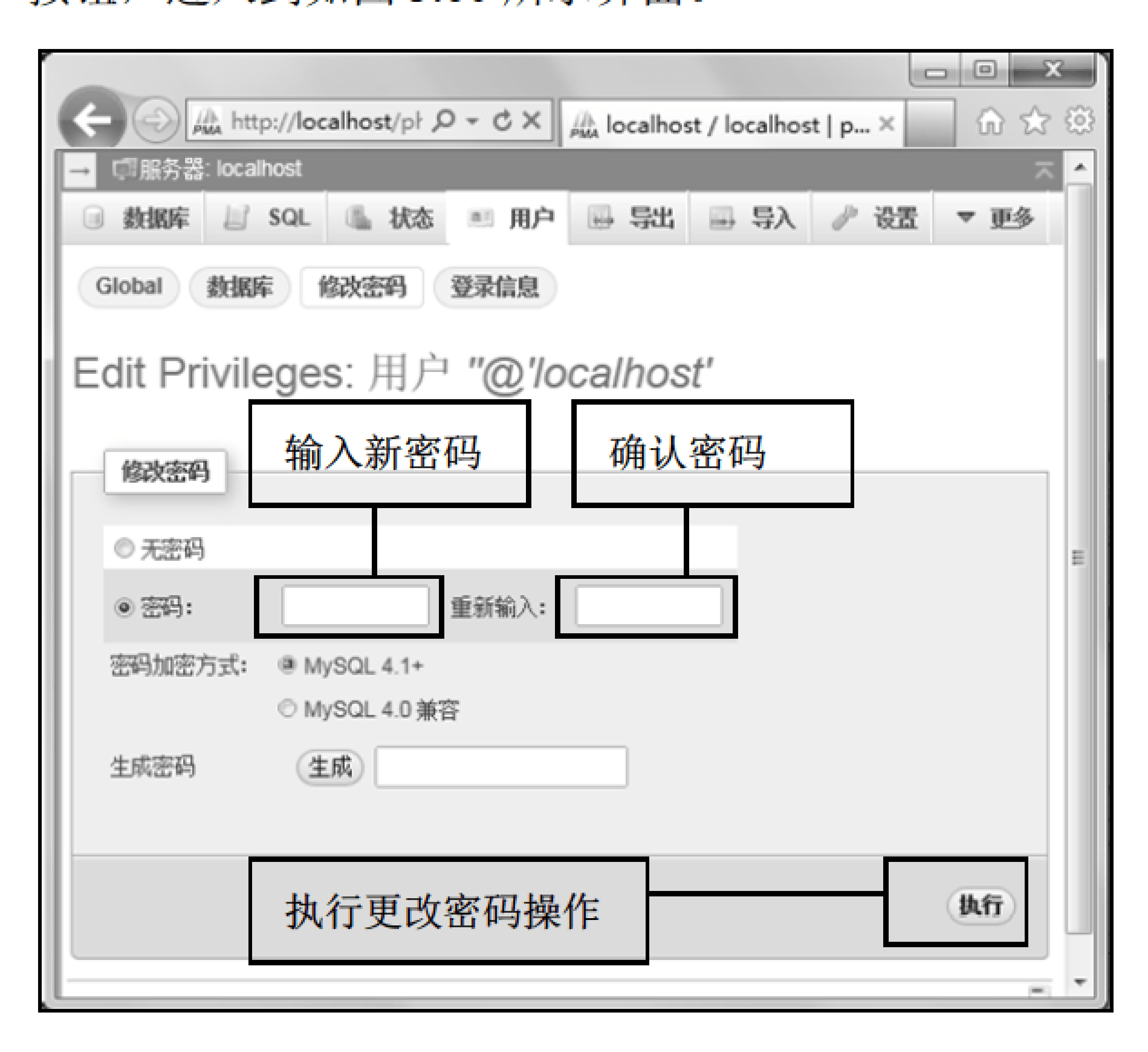


图 3.60　修改密码

在如图 3.60 所示的界面中，可以设置用户的权限、修改密码、更改登录用户信息和复制用户。在输入新密码和确认密码之后，单击“执行”按钮，完成对用户密码的修改操作，返回主页面，将提示密码修改成功。

3.3 小　　结

本章主要介绍了两个 MySQL 的图形化管理工具，一个是 MySQL 官网中提供的 MySQL Workbench，另一个是使用 PHP 开发的 phpMyAdmin，通过这些图形化管理软件可以很方便地操作 MySQL 数据库，其中包括创建数据库/数据表、管理数据库/数据表，以及数据的导入和导出等都是十分方便的。对于这两个图形化管理工具，读者可根据自己的喜好来进行选择，使用哪个都可以。

3.4 实践与练习

1．尝试使用 MySQL Workbench 图形化管理工具创建一个名称为 db_mydatabase 的数据库，并在该数据库中添加一名称为 tb_user 的数据表。

2．尝试使用 phpMyAdmin 图形化管理工具创建一个名称为 db_mydatabase_1 的数据库，并在该数据库中添加一名称为 tb_user 的数据表。

第 4 章

数据库操作

（视频讲解：6 分钟）

启动并连接 MySQL 服务器后，即可对 MySQL 数据库进行操作，操作 MySQL 数据库的方法非常简单。本章中将对操作 MySQL 数据库中的创建数据库、修改数据库、查看数据库、选择数据库和删除数据库进行详细介绍。

通过阅读本章，读者可以：

- 了解数据库的相关知识
- 掌握创建数据库的方法
- 了解查看数据库的方法
- 了解选择数据库的方法
- 掌握修改数据库的方法
- 掌握删除数据库的方法

4.1 认识数据库

在进行数据库操作前，首先需要对其有一个基本的了解。本节将对数据库的基本概念、数据库对象及其相关知识进行详细的介绍。

4.1.1 数据库基本概念

数据库（DataBase）是按照数据结构来组织、存储和管理数据的仓库，是存储在一起的相关数据的集合。其优点主要体现在以下几方面。

（1）减少数据的冗余度，节省数据的存储空间；

（2）具有较高的数据独立性和易扩充性；

（3）实现数据资源的充分共享。

下面介绍与数据库相关的几个概念。

1. 数据库系统

数据库系统（DataBase System，DBS）是采用数据库技术的计算机系统，是由数据库（数据）、数据库管理系统（软件）、数据库管理员（人员）、硬件平台（硬件）和软件平台（软件）5 部分构成的运行实体。其中，数据库管理员（DataBase Administrator，DBA）是对数据库进行规划、设计、维护和监视等的专业管理人员，在数据库系统中起着非常重要的作用。

2. 数据库管理系统

数据库管理系统（DataBase Management System，DBMS）是数据库系统的一个重要组成部分，是位于用户与操作之间的一层数据管理软件，负责数据库中的数据组织、数据操纵、数据维护和数据服务等。主要具有如下功能。

（1）数据存取的物理构建：为数据模式的物理存取与构建提供有效的存取方法与手段。

（2）数据操纵功能：为用户使用数据库的数据提供方便，如查询、插入、修改、删除等以及简单的算术运算和统计。

（3）数据定义功能：用户可以通过数据库管理系统提供的数据定义语言（Data Definition Language，DDL）方便地对数据库中的对象进行定义。

（4）数据库的运行管理：数据库管理系统统一管理数据库的运行和维护，以保障数据的安全性、完整性、并发性和故障的系统恢复性。

（5）数据库的建立和维护功能：数据库管理系统能够完成初始数据的输入和转换、数据库的转储和恢复、数据库的性能监视和分析等任务。

3. 关系数据库

关系数据库是支持关系模型的数据库。关系模型由关系数据结构、关系操作集合和完整性约束 3



部分组成。

（1）关系数据结构：在关系模型中数据结构单一，现实世界的实体以及实体间的联系均用关系来表示，实际上关系模型中数据结构就是一张二维表。

（2）关系操作集合：关系操作分为关系代数、关系演算、具有关系代数和关系演算双重特点的语言（SQL）。

（3）完整性约束：完整性约束包括实体完整性、参照完整性和用户定义完整性。

4.1.2　数据库常用对象

在 MySQL 的数据库中，表、视图、存储过程和索引等具体存储数据或对数据进行操作的实体都被称为数据库对象。下面介绍几种常用的数据库对象。

1. 表

表是包含数据库中所有数据的数据库对象，由行和列组成，用于组织和存储数据。

2. 字段

表中每列称为一个字段，字段具有自己的属性，如字段类型、字段大小等。其中，字段类型是字段最重要的属性，它决定了字段能够存储哪种数据。

SQL 规范支持 5 种基本字段类型：字符型、文本型、数值型、逻辑型和日期时间型。

3. 索引

索引是一个单独的、物理的数据库结构。它是依赖于表建立的，在数据库中索引使数据库程序无须对整个表进行扫描，就可以在其中找到所需的数据。

4. 视图

视图是从一张或多张表中导出的表（也称虚拟表），是用户查看数据表中数据的一种方式。表中包括几个被定义的数据列与数据行，其结构和数据建立在对表的查询基础之上。

5. 存储过程

存储过程（Stored Procedure）是一组为了完成特定功能的 SQL 语句集合（包含查询、插入、删除和更新等操作），经编译后以名称的形式存储在 SQL Server 服务器端的数据库中，由用户通过指定存储过程的名字来执行。当这个存储过程被调用执行时，这些操作也会同时执行。

4.1.3　系统数据库

系统数据库是指安装完 MySQL 服务器后，会附带一些数据库。例如，在默认安装的 MySQL Workbench 中，会默认创建如图 4.1 所示的 5 个数据库，这些数据库就称为系统数据库。系统数据库会记录一些必需的信息，用户是不能直接修改这些系统数据库的。test 和 sakila 除外，这两个数据库中的

信息对于系统不是必需的，可以进行修改。下面将对图 4.1 中所列的系统数据库分别进行介绍。

1. information_schema 数据库

information_schema 数据库主要用于存储数据库对象的相关信息。例如，用户表信息、列信息、权限信息、字符集信息和分区信息等。

2. performance_schema 数据库

performance_schema 数据库主要用于存储数据库服务器性能参数。

3. sakila 数据库

sakila 数据库是 MySQL 提供的样例数据库。该数据库共有 16 张数据表，这些数据表都是比较常见的，在设计数据库时，可以参照这些样例数据表来快速完成所需的数据表。

4. test 数据库

test 数据库是 MySQL 数据库管理系统自动创建的测试数据库，该数据库中没有创建任何数据表，对于任何用户都可以使用这个数据库。一般情况下，不建议直接使用该数据库。

5. world 数据库

world 数据库是 MySQL 数据库管理系统自动创建的数据库，该数据库中只包括 3 张数据表，分别保存城市、国家和国家使用的语言等内容。

说明

在 MySQL 中，使用最多的是用户数据库，也就是用户根据实际需求所创建的数据库，例如在图 4.2 中的 db_mrsoft 就是我们创建的用户数据库。

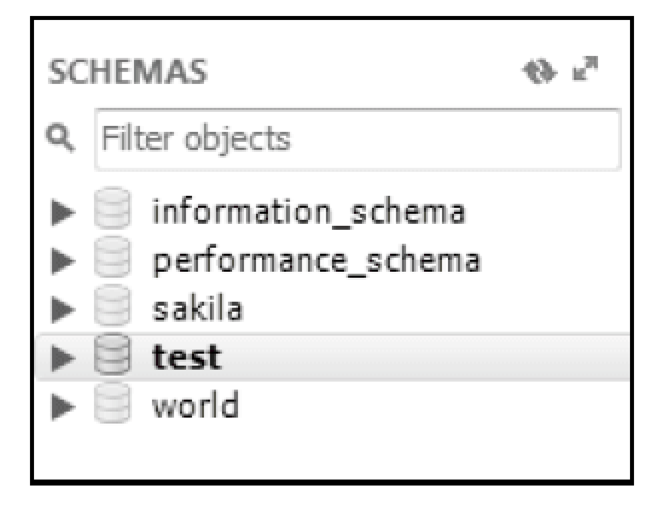


图 4.1　系统数据库

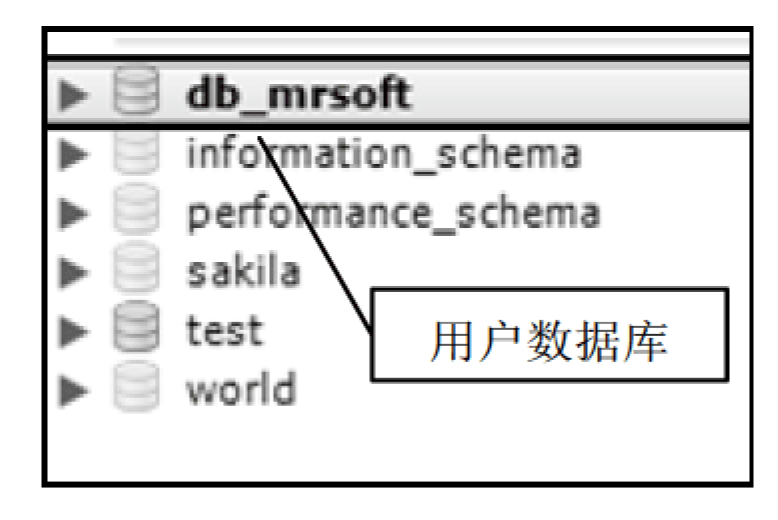


图 4.2　用户数据库

4.2　创建数据库

在 MySQL 中，可以使用 CREATE DATABASE 语句和 CREATE SCHEMA 语句创建 MySQL 数据



库，其语法如下。

```
CREATE  {DATABASE|SCHEMA}  [IF NOT EXISTS] 数据库名
[
    [DEFAULT] CHARACTER SET [=]  字符集  |
    [DEFAULT] COLLATE [=]  校对规则名称
];
```



在语法中，花括号“{}”表示必选项；中括号“[]”表示可选项；竖线“|”表示分隔符两侧的内容为“或”的关系。在上面的语法中，{DATABASE|SCHEMA}表示要么使用关键字 DATABASE，要么使用 SCHEMA，但不能全不使用。

参数说明如下。

（1）[IF NOT EXISTS]：可选项，表示在创建数据库前进行判断，只有该数据库目前尚未存在时才执行创建语句。

（2）数据库名：必须指定的，在文件系统中，MySQL 的数据存储区将以目录方式表示 MySQL 数据库。因此，这里的数据库名必须符合操作系统文件夹的命名规则，而在 MySQL 中是不区分大小写的。

（3）[DEFAULT]：可选项，表示指定默认值。

（4）CHARACTER SET [=] 字符集：可选项，用于指定数据库的字符集。如果不想指定数据库所使用的字符集，那么就可以不使用该项，这时 MySQL 会根据 MySQL 服务器默认使用的字符集来创建该数据库。这里的字符集可以是 GB2312 或者 GBK（简体中文）、UTF8（针对 Unicode 的可变长度的字符编码，也称万国码）、BIG5（繁体中文）、Latin1（拉丁文）等。其中最常用的就是 UTF8 和 GBK。

（5）COLLATE [=] 校对规则名称：可选项，用于指定字符集的校对规则。例如，utf8_bin 或者 gbk_chinese_ci。具体都有哪些校对规则，可以在 MySQL 的图形化工具 Workbench 的创建数据库的窗口中找到，如图 4.3 所示。

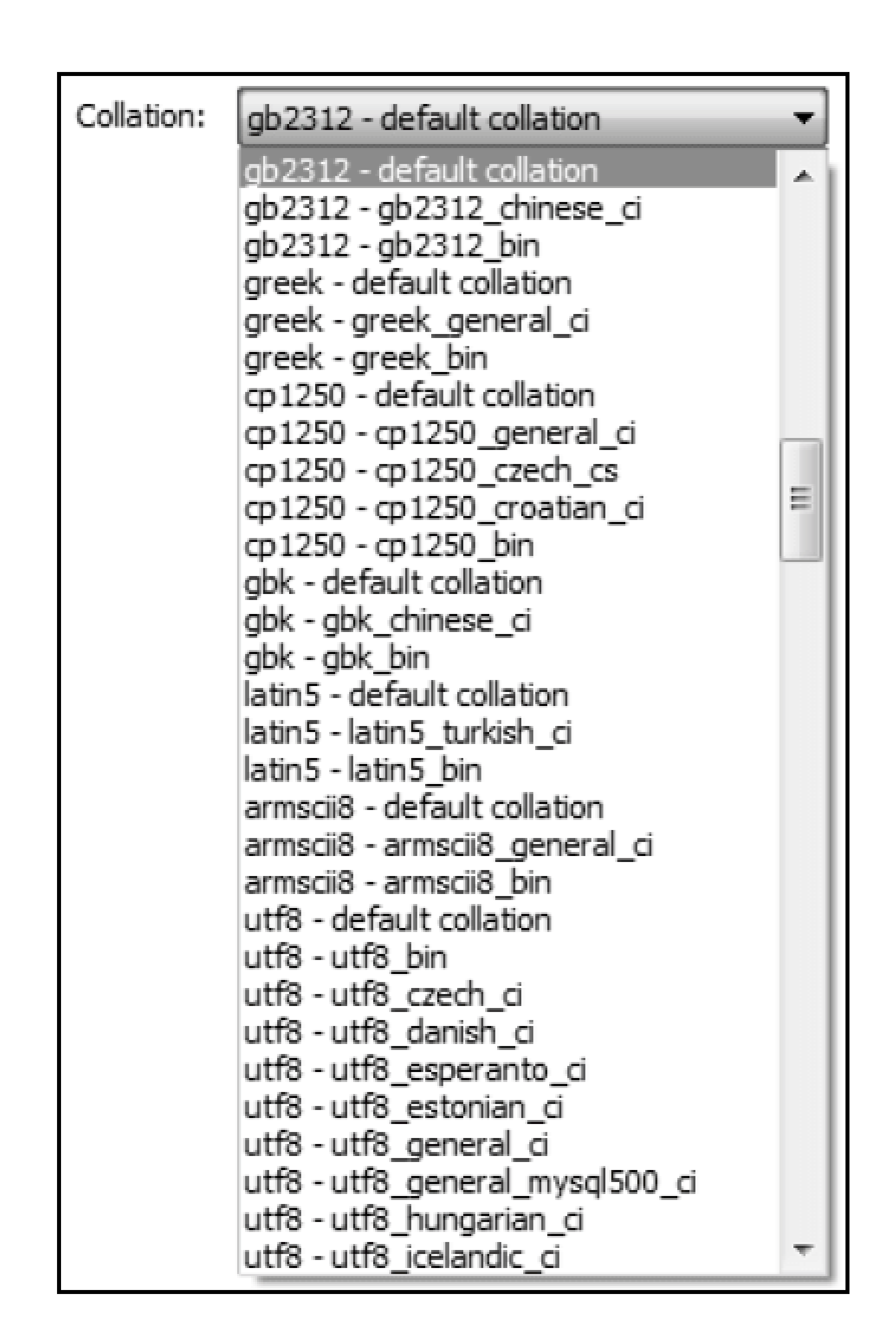


图 4.3　字符集的校对规则

在创建数据库时，数据库命名有以下几项规则。

（1）不能与其他数据库重名，否则将发生错误。

（2）名称可以由任意字母、阿拉伯数字、下划线（_）和“$”组成，可以使用上述的任意字符开头，但不能使用单独的数字，否则会造成它与数值相混淆。

（3）名称最长可为 64 个字符，而别名最多可长达 256 个字符。

（4）不能使用 MySQL 关键字作为数据库名、表名。

（5）默认情况下，在 Windows 下数据库名、表名的大小写是不敏感的，而在 Linux 下数据库名、

表名的大小写是敏感的。为了便于数据库在平台间进行移植，建议读者采用小写来定义数据库名和表名。

4.2.1　通过 CREATE DATABASE 语句创建基本数据库

例 4.1　通过 CREATE DATABASE 语句创建一个名称为 db_admin 的数据库，具体代码如下。**（实例位置：光盘\TM\sl\4\4.1）**

```
create database db_admin;
```

运行效果如图 4.4 所示。

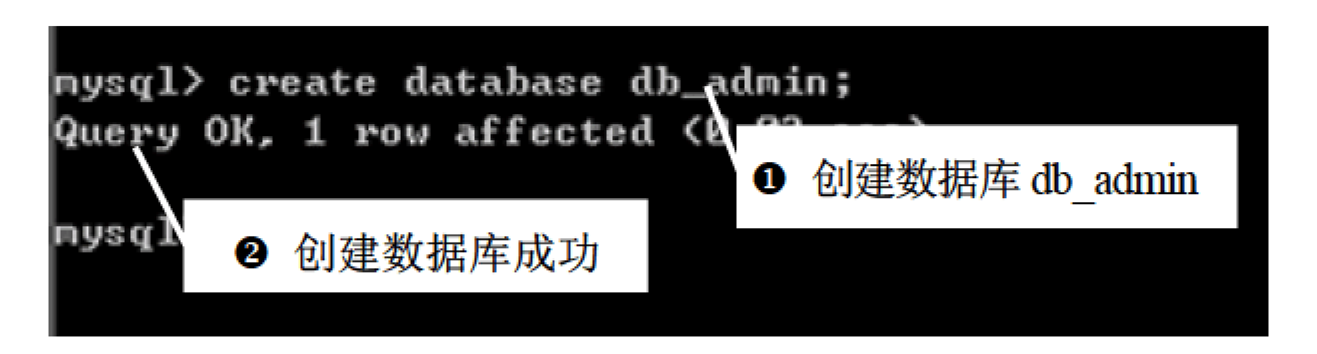


图 4.4　通过 CREATE DATABASE 语句创建 MySQL 数据库

4.2.2　通过 CREATE SCHEMA 语句创建基本数据库

上面介绍的例 4.1 是最基本的创建数据库的方法，实际上，还可以通过语法中给出的 CREATE SCHEMA 来创建数据库，两者的功能是一样的。在使用 MySQL 官网中提供的 MySQL Workbench 图形化工具创建数据库时，使用的就是这种方法。

例 4.2　通过 CREATE SCHEMA 语句创建一个名称为 db_admin1 的数据库，具体代码如下。**（实例位置：光盘\TM\sl\4\4.2）**

```
create schema db_admin1;
```

运行效果如图 4.5 所示。

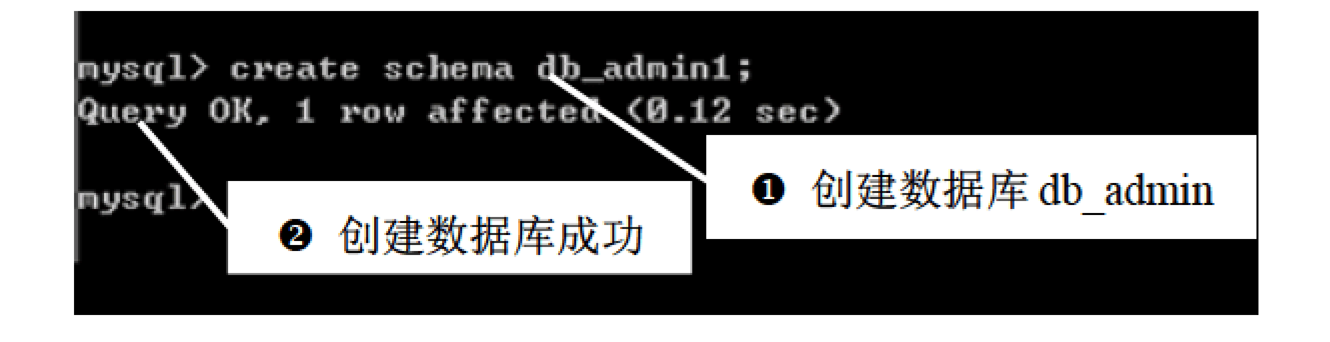
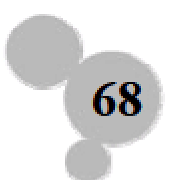


图 4.5　通过 CREATE SCHEMA 语句创建 MySQL 数据库

4.2.3　创建指定字符集的数据库

在创建数据库时，如果不指定其使用的字符集或者是字符集的校对规则，那么将根据 my.ini 文件



中指定的 default-character-set 变量的值来设置其使用的字符集。从创建数据库的基本语法中可以看出，在创建数据库时，还可以指定数据库所使用的字符集，下面将通过一个具体的例子来演示如何在创建数据库时，指定字符集。

例 4.3　通过 CREATE DATABASE 语句创建一个名称为 db_test 的数据库，并指定其字符集为 GBK，具体代码如下。**（实例位置：光盘\TM\sl\4\4.3）**

```
CREATE DATABASE db_test
CHARACTER SET = GBK;
```

运行效果如图 4.6 所示。

```
mysql> CREATE DATABASE db_test
    -> CHARACTER SET = GBK;
Query OK, 1 row affected (0.03 sec)

mysql> _
```

设置字符集为 GBK

图 4.6　创建使用 GBK 字符集的 MySQL 数据库

4.2.4　创建数据库前判断是否存在同名数据库

在 MySQL 中，不允许同一系统中存在两个相同名称的数据库，如果要创建的数据库名称已经存在，那么系统将给出以下错误信息。

```
ERROR 1007 (HY000): Can't create database 'db_test'; database exists
```

为了避免错误的发生，在创建数据库时，可以使用 IF NOT EXISTS 选项来实现在创建数据库前判断该数据库是否存在，只有不存在时才会进行创建。

例 4.4　通过 CREATE DATABASE 语句创建一个名称为 db_test1 的数据库，并在创建前判断该数据库名称是否存在，只有不存在时才进行创建，具体代码如下。**（实例位置：光盘\TM\sl\4\4.4）**

```
CREATE DATABASE IF NOT EXISTS db_test1;
```

运行效果如图 4.7 所示。

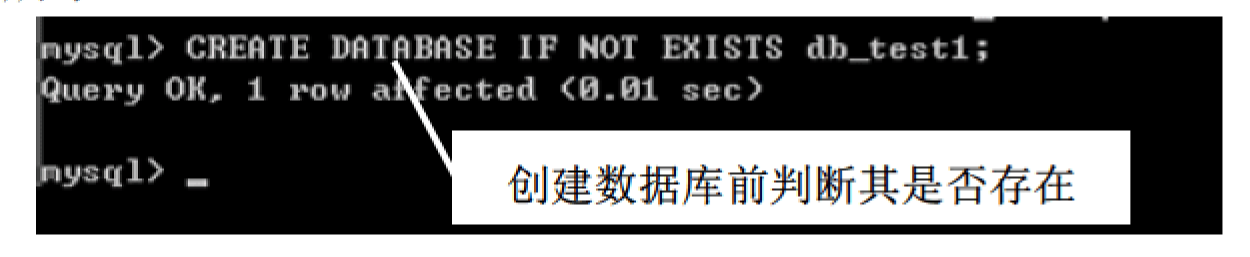


图 4.7　创建数据库前判断是否存在同名数据库

再次执行上面的语句，将不再创建数据库 db_test1，显示效果如图 4.8 所示。



图 4.8　创建已经存在的数据库的效果

4.3　查看数据库

成功创建数据库后，可以使用 SHOW 命令查看 MySQL 服务器中的所有数据库信息，语法如下。

```
SHOW  {DATABASES|SCHEMAS}
[LIKE '模式' WHERE 条件]
;
```

参数说明如下。

（1）{DATABASES|SCHEMAS}：表示必须有一个是必选项，用于列出当前用户权限范围内所能查看到的所有数据库名称。这两个选项的结果是一样的，使用哪个都可以。

（2）LIKE：可选项，用于指定匹配模式。

（3）WHERE：可选项，用于指定数据库名称查询范围的条件。

例 4.5　在 4.2.1 节中创建了数据库 db_admin，下面使用 SHOW DATABASES 语句查看 MySQL 服务器中的所有数据库名称，代码如下。**（实例位置：光盘\TM\sl\4\4.5）**

```
SHOW DATABASES;
```

运行结果如图 4.9 所示。

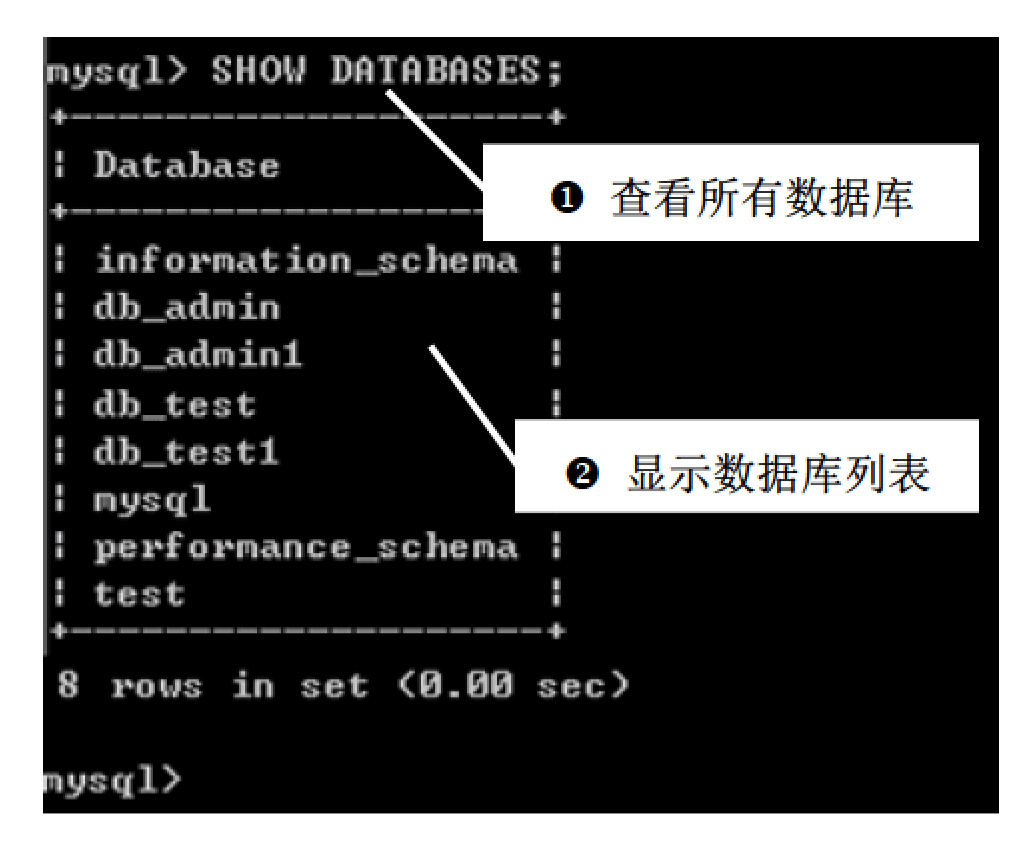


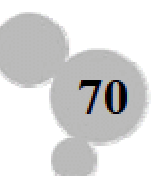
图 4.9　查看数据库

从图 4.9 运行的结果可以看出，通过 SHOW 命令查看 MySQL 服务器中的所有数据库，结果显示 MySQL 服务器中有 8 个数据库，这 8 个数据库包括系统数据库。

如果 MySQL 服务器中的数据库比较多，也可以通过指定匹配模式来筛选想要得到的数据库，下面将通过一个具体的实例来演示如何通过 LIKE 关键字筛选要查看的数据库。

例 4.6　筛选以 db_开头的数据库名称，代码如下。**（实例位置：光盘\TM\sl\4\4.6）**

```
SHOW DATABASES LIKE 'db_%';
```



执行效果如图 4.10 所示。

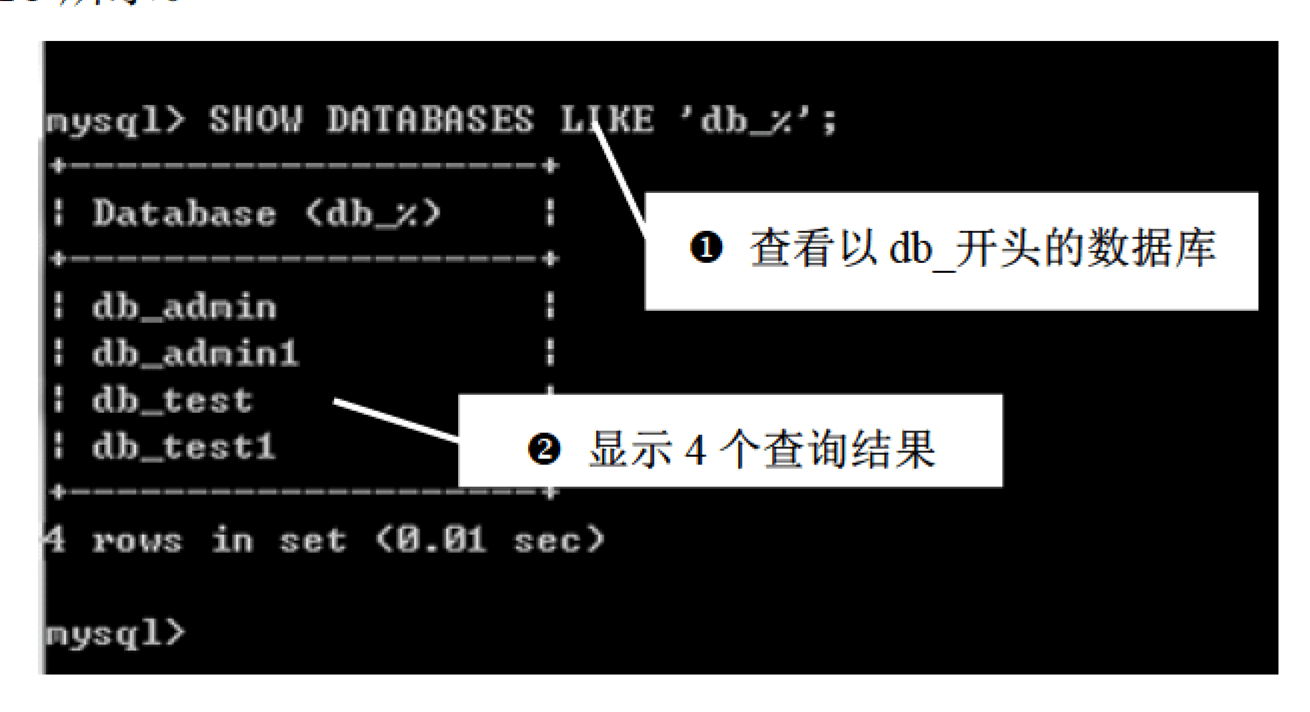


图 4.10　筛选以 db_开头的数据库名称

4.4　选择数据库

在 MySQL 中，使用 CREATE DATABASE 语句创建数据库后，该数据库并不会自动成为当前数据库。如果想让它成为当前数据库，需要使用 MySQL 提供的 USE 语句来实现，USE 语句可以实现选择一个数据库，使其成为当前数据库。只有使用 USE 语句指定某个数据库为当前数据库后，才能对该数据库及其存储的数据对象执行操作。USE 语句的语法格式如下。

```
USE　数据库名;
```



使用 USE 语句将数据库指定为当前数据库后，当前数据库在当前工作会话关闭（即断开与该数据库的连接）或再次使用 USE 语句指定数据库时，结束工作状态。

例 4.7　选择名称为 db_admin 的数据库，设置其为当前默认的数据库，具体代码如下。**（实例位置：光盘\TM\sl\4\4.7）**

```
USE db_admin;
```

执行结果如图 4.11 所示。



图 4.11　选择数据库

4.5 修改数据库

在 MySQL 中，创建一个数据库后，还可以对其进行修改，不过这里的修改是指可以修改被创建数据库的相关参数，并不能修改数据库名。修改数据库名不能使用这个语句。修改数据库可以使用 ALTER DATABASE 或者 ALTER SCHEMA 语句来实现。修改数据库的语句的语法格式如下。

```
ALTER {DATABASE | SCHEMA} [数据库名]
 [DEFAULT] CHARACTER SET [=] 字符集
| [DEFAULT] COLLATER [=] 校对规则名称
```

参数说明如下。

（1）{DATABASES|SCHEMAS}：表示必须有一个是必选项，这两个选项的结果是一样的，使用哪个都可以。

（2）[数据库名]：可选项，如果不指定要修改的数据库，那么将表示修改当前（默认）的数据库。

（3）[DEFAULT]：可选项，表示指定默认值。

（4）CHARACTER SET [=] 字符集：可选项，用于指定数据库的字符集。如果不想指定数据库所使用的字符集，那么就可以不使用该项，这时 MySQL 会根据 MySQL 服务器默认使用的字符集来创建该数据库。这里的字符集可以是 GB2312 或者 GBK（简体中文）、UTF8（针对 Unicode 的可变长度的字符编码，也称万国码）、BIG5（繁体中文）、Latin1（拉丁文）等。其中最常用的就是 UTF8 和 GBK。

（5）COLLATE [=] 校对规则名称：可选项，用于指定字符集的校对规则。例如，utf8_bin 或者 gbk_chinese_ci。这与 4.2 节创建数据库的语法中的该从句是相同的。

注意

在使用 ALTER DATABASE 或者 ALTER SCHEMA 语句时，用户必须具有对数据库进行修改的权限。

例 4.8 修改例 4.1 中创建的数据库 db_admin，设置默认字符集和校对规则，具体代码如下。**（实例位置：光盘\TM\sl\4\4.8）**

```
ALTER DATABASE db_admin
      DEFAULT CHARACTER SET gbk
      DEFAULT COLLATE gbk_chinese_ci;
```

执行结果如图 4.12 所示。





图 4.12　设置默认字符集和校对规则

4.6　删除数据库

在 MySQL 中，可以通过使用 DROP DATABASE 语句或者 DROP SCHEMA 语句来删除已经存在的数据库。使用该命令删除数据库的同时，该数据库中的表，以及表中的数据也将永久删除，因此，在使用该语句删除数据库时一定要小心，以免误删除有用的数据库。DROP DATABASE 或者 DROP SCHEMA 语句的语法格式如下。

```
DROP {DATABASE|SCHEMA} [IF EXISTS]  数据库名;
```

参数说明如下。

（1）{DATABASES|SCHEMAS}：表示必须有一个是必选项，这两个选项的结果是一样的，使用哪个都可以。

（2）[IF EXISTS]：用于指定在删除数据前，先判断该数据库是否已经存在，只有已经存在时，才会执行删除操作，这样可以避免删除不存在的数据库时，产生异常。

注意

在使用 DROP DATABASE 或者 DROP SCHEMA 语句时，用户必须具有对数据库进行删除的权限。

注意

在删除数据库时，该数据库上的用户权限是不会自动被删除的。

注意

删除数据库的操作应该谨慎使用，一旦执行该操作，数据库的所有结构和数据都会被删除，没有恢复的可能，除非数据库有备份。

例 4.9　通过 DROP DATABASE 语句删除名为 db_admin 的数据库，具体代码如下。**（实例位置：光盘\TM\sl\4\4.9）**

```
DROP DATABASE db_admin;
```

执行效果如图 4.13 所示。



图 4.13　删除数据库

当使用上面的命令删除数据库时，如果指定的数据库不存在，将产生如图 4.14 所示的异常信息。

```
mysql> DROP DATABASE db_111;
ERROR 1008 (HY000): Can't drop database 'db_111'; database doesn't exist
mysql> _
```

图 4.14　删除不存在的数据库出错

为了解决这一问题，可以在 DROP DATABASE 语句中使用 IF EXISTS 从句来保证只有当数据库存在时才执行删除数据库的操作。下面通过一个具体的例子来演示这一功能。

例 4.10　通过 DROP DATABASE 语句删除名称为 db_111 的数据库（该数据库不存在），具体代码如下。**（实例位置：光盘\TM\sl\4\4.10）**

```
SHOW DATABASES LIKE 'db_%';
DROP DATABASE IF EXISTS db_111;
```

执行效果如图 4.15 所示。

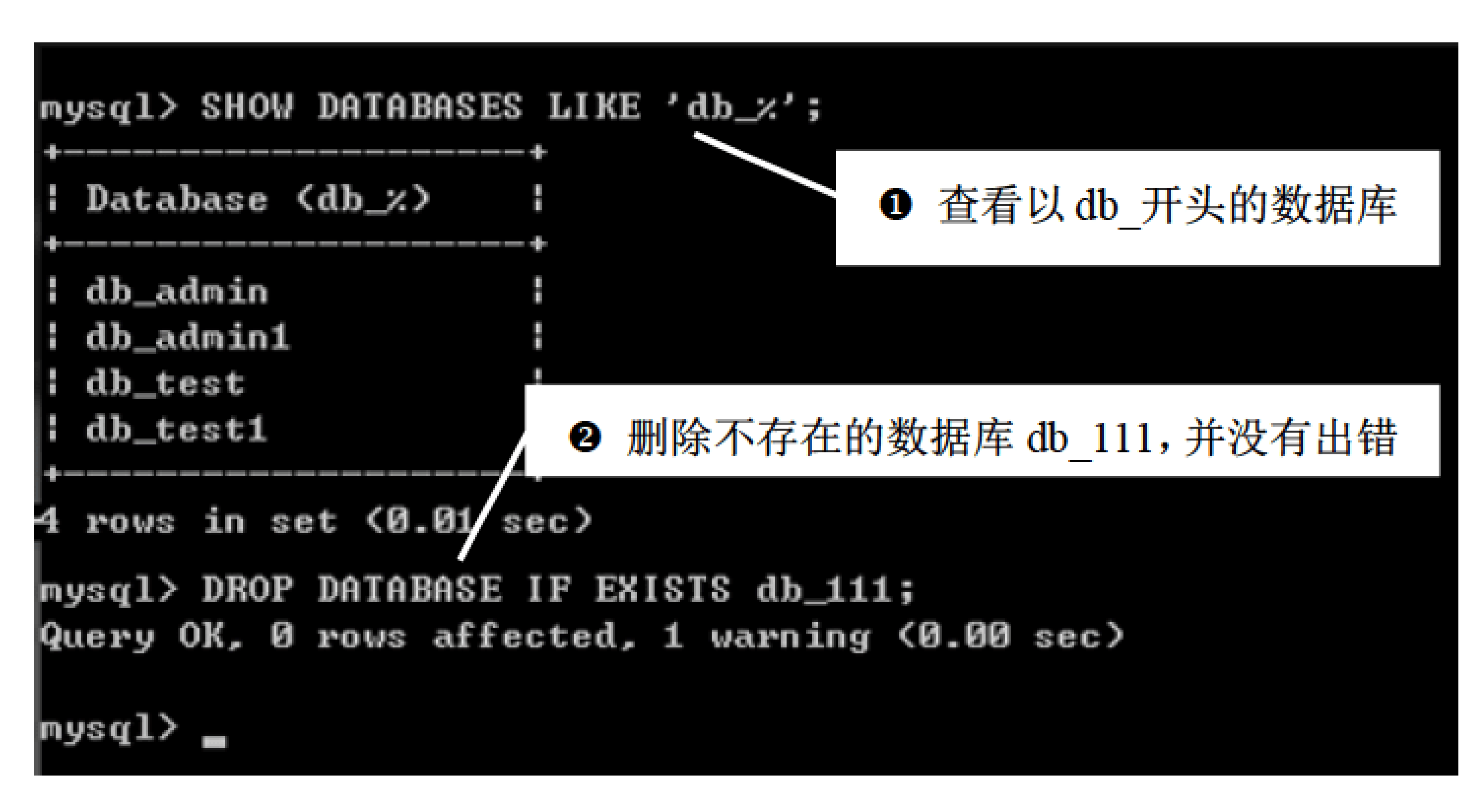


图 4.15　删除不存在的数据库未出错

注意

MySQL 安装后，系统会自动创建两个名称分别为 performance_schema 和 mysql 的系统数据库，MySQL 把与数据库相关的信息存储在这两个系统数据库中，如果删除了这两个数据库，那么 MySQL 将不能正常工作，所以这两个数据库一定不能删除。

4.7　小　　结

本章首先介绍了数据库的基本概念、数据库的常用对象，以及 MySQL 中的系统数据库，然后介绍了如何修改数据库、查看数据库、选择数据库、修改数据库和删除数据库。其中，创建数据库、选择数据库和删除数据库需要重点掌握，在实际开发中经常会应用到。

4.8　实践与练习

1．通过 CREATE SCHEMA 语句创建一个名称为 db_mr 的数据库，并指定其字符集为 UTF8。**（答案位置：光盘\TM\sl\4\4.11）**

2．通过 DROP SCHEMA 语句删除第 1 题中创建的数据库 db_mr，并且指定只有该数据库存在时才删除。**（答案位置：光盘\TM\sl\4\4.12）**

3．通过 SHOW SCHEMAS 语句筛选以 db_开头的数据库名称。**（答案位置：光盘\TM\sl\4\4.13）**

第 5 章

存储引擎及数据类型

（视频讲解：12 分钟）

使用存储引擎可以加快查询的速度，并且每一种引擎都存在不同的含义。MySQL 的数据类型是数据的一种属性，其可以决定数据的存储格式、有效范围和相应的限制，并且可以让读者了解如何选择合适的数据类型。本章将对 MySQL 的存储引擎和数据类型的使用进行详细的讲解。

通过阅读本章，读者可以：

- 了解 MySQL 存储引擎
- 了解并查询 MySQL 中支持的存储引擎
- 掌握选择存储引擎的方法
- 掌握设置数据表的存储引擎的方法
- 掌握 MySQL 的数据类型

5.1　MySQL 存储引擎

存储引擎其实就是如何存储数据、如何为存储的数据建立索引和如何更新、查询数据等技术的实现方法。因为在关系数据库中数据是以表的形式存储的，所以存储引擎也可以称为表类型（即存储和操作此表的类型）。在 Oracle 和 SQL Server 等数据库中只有一种存储引擎，所有数据存储管理机制都是一样的；而 MySQL 数据库提供了多种存储引擎。用户可以根据不同的需求为数据表选择不同的存储引擎，用户也可以根据需要编写自己的存储引擎。

5.1.1　MySQL 存储引擎的概念

MySQL 中的数据用各种不同的技术存储在文件（或者内存）中。这些技术中的每一种技术都使用不同的存储机制、索引技巧、锁定水平并且最终提供广泛的、不同的功能和能力。通过选择不同的技术，能够获得额外的速度或者功能，从而改善应用的整体功能。

这些不同的技术以及配套的相关功能在 MySQL 中被称作存储引擎（也称作表类型）。MySQL 默认配置了许多不同的存储引擎，可以预先设置或者在 MySQL 服务器中启用。可以选择适用于服务器、数据库和表格的存储引擎，以便在选择如何存储信息、如何检索这些信息以及需要的数据结合什么性能和功能的时候为其提供最大的灵活性。

5.1.2　查询 MySQL 中支持的存储引擎

1．查询支持的全部存储引擎

在 MySQL 中，可以使用 SHOW ENGINES 语句查询 MySQL 中支持的存储引擎。其查询语句如下。

```
SHOW ENGINES;
```

SHOW ENGINES 语句可以用“;”结束，也可以用“\g”或者“\G”结束。“\g”与“;”的作用是相同的，“\G”可以让结果显示得更加美观。

使用 SHOW ENGINES \g 语句查询的结果如图 5.1 所示。

```
mysql> show engines\g
+------------+---------+----------------------------------------------------------------
--------+--------------+------+------------+
| Engine     | Support | Comment
        | Transactions | XA   | Savepoints |
+------------+---------+----------------------------------------------------------------
--------+--------------+------+------------+
| MyISAM     | YES     | Default engine as of MySQL 3.23 with great performance
        | NO           | NO   | NO         |
| CSV        | YES     | CSV storage engine
        | NO           | NO   | NO         |
| MRG_MYISAM | YES     | Collection of identical MyISAM tables
        | NO           | NO   | NO         |
| BLACKHOLE  | YES     | /dev/null storage engine (anything you write to it disa
ppears) | NO           | NO   | NO         |
| FEDERATED  | NO      | Federated MySQL storage engine
        | NULL         | NULL | NULL       |
| InnoDB     | DEFAULT | Supports transactions, row-level locking, and foreign k
eys     | YES          | YES  | YES        |
| ARCHIVE    | YES     | Archive storage engine
        | NO           | NO   | NO         |
| MEMORY     | YES     | Hash based, stored in memory, useful for temporary tabl
es      | NO           | NO   | NO         |
+------------+---------+----------------------------------------------------------------
--------+--------------+------+------------+
8 rows in set (0.06 sec)
```

图 5.1　使用 SHOW ENGINES \G 语句查询 MySQL 中支持的存储引擎

使用 SHOW ENGINES \G 语句查询的结果如图 5.2 所示。

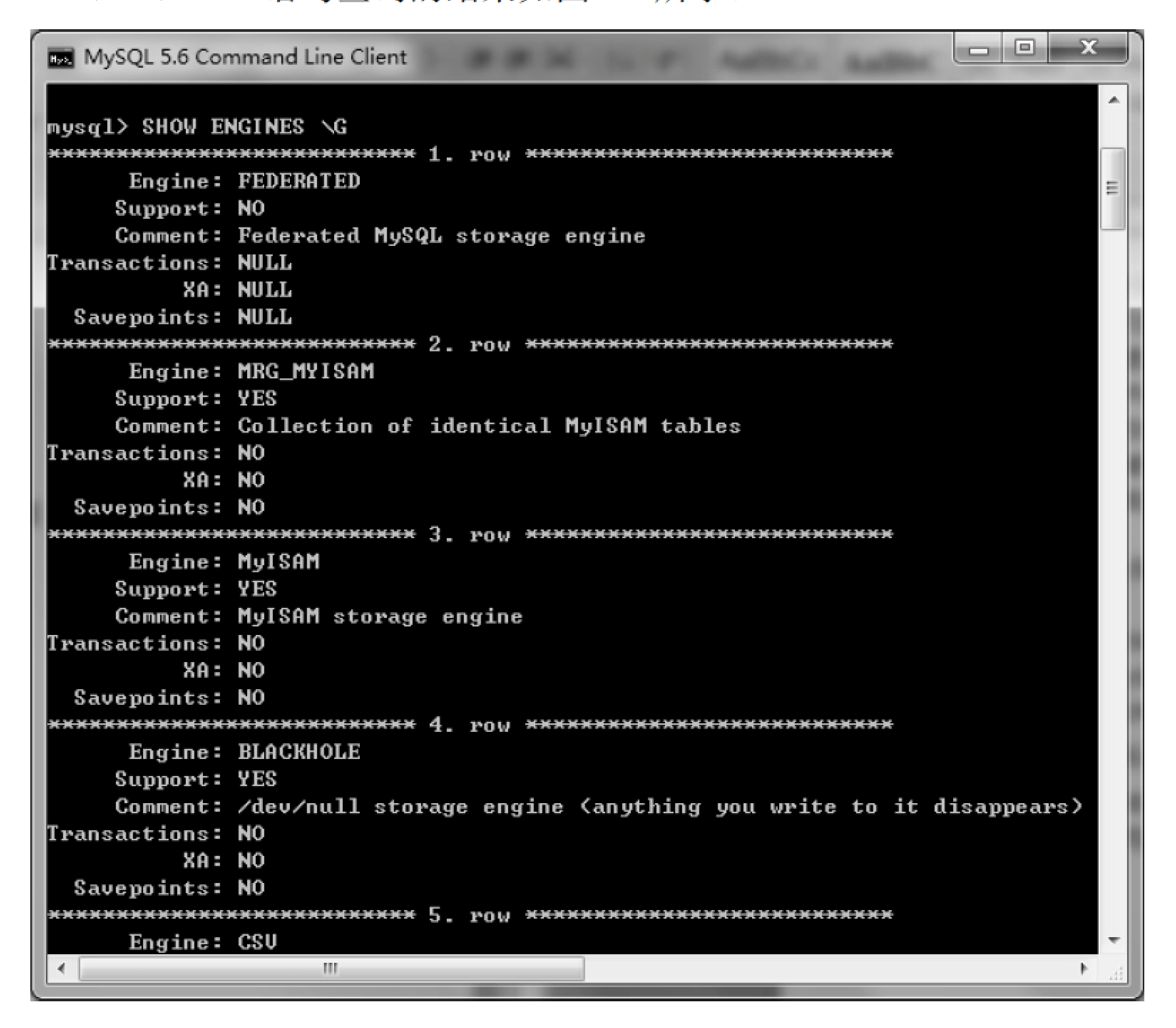


图 5.2　使用 SHOW ENGINES \G 语句查询 MySQL 中支持的存储引擎

查询结果中的 Engine 参数指的是存储引擎的名称；Support 参数指的是 MySQL 是否支持该类引擎，YES 表示支持；Comment 参数指对该引擎的评论。

从查询结果中可以看出，MySQL 支持多个存储引擎，其中 InnoDB 为默认存储引擎。

2. 查询默认的存储引擎

如果想要知道当前 MySQL 服务器采用的默认存储引擎是什么，可以通过执行 SHOW VARIABLES 命令来查看。查询默认的存储引擎的 SQL 语句如下。

```
SHOW VARIABLES LIKE 'storage_engine%';
```

例 5.1　查询默认的存储引擎，具体代码如下。**（实例位置：光盘\TM\sl\5\5.1）**

```
SHOW VARIABLES LIKE 'storage_engine%';
```

执行效果如图 5.3 所示。

```
mysql> SHOW VARIABLES LIKE 'storage_engine%';
+----------------+--------+
| Variable_name  | Value  |
+----------------+--------+
| storage_engine | InnoDB |
+----------------+--------+
1 row in set (0.00 sec)

mysql>
```

图 5.3　查询默认的存储引擎

从图 5.3 中可以看出，当前 MySQL 服务器采用的默认存储引擎是 InnoDB。

有些表根本不用来存储长期数据，实际上用户需要完全在服务器的 RAM 或特殊的临时文件中创建和维护这些数据，以确保高性能，但这样也存在很高的不稳定风险。还有一些表只是为了简化对一组相同表的维护和访问，为同时与所有这些表交互提供一个单一接口。另外，还有其他一些特别用途的表，但重点是：MySQL 支持很多类型的表，每种类型都有自己特定的作用、优点和缺点。MySQL 还相应地提供了很多不同的存储引擎，可以以最适合于应用需求的方式存储数据。MySQL 有多个可用的存储引擎，下面主要介绍 InnoDB、MyISAM 和 MEMORY3 种存储引擎。

5.1.3　InnoDB 存储引擎

InnoDB 已经开发了十余年，遵循 CNU 通用公开许可（GPL）发行。InnoDB 已经被一些重量级 Internet 公司所采用，如雅虎、Slashdot 和 Google，为用户操作非常大的数据库提供了一个强大的解决方案。InnoDB 给 MySQL 的表提供了事务、回滚、崩溃修复能力和多版本并发控制的事务安全。在 MySQL 从 3.23.34a 开始包含 InnoDB 存储引擎。InnoDB 是 MySQL 上第一个提供外键约束的表引擎。而且 InnoDB 对事务处理的能力，也是 MySQL 其他存储引擎所无法与之比拟的。下面介绍 InnoDB 存储引擎的特点及其优缺点。

InnoDB 存储引擎中支持自动增长列 AUTO_INCREMENT。自动增长列的值不能为空，且值必须

唯一。MySQL 中规定自增列必须为主键。在插入值时，如果自动增长列不输入值，则插入的值为自动增长后的值；如果输入的值为 0 或空（NULL），则插入的值也为自动增长后的值；如果插入某个确定的值，且该值在前面没有出现过，则可以直接插入。

InnoDB 存储引擎中支持外键（FOREIGN KEY）。外键所在的表为子表，外键所依赖的表为父表。父表中被子表外键关联的字段必须为主键。当删除、更新父表的某条信息时，子表也必须有相应的改变。InnoDB 存储引擎中，创建的表的表结构存储在.frm 文件中。数据和索引存储在 innodb_data_home_dir 和 innodb_data_file_path 表空间中。

InnoDB 存储引擎的优势在于提供了良好的事务管理、崩溃修复能力和并发控制。缺点是其读写效率稍差，占用的数据空间相对比较大。

InnoDB 表是如下情况的理想引擎。

（1）更新密集的表：InnoDB 存储引擎特别适合处理多重并发的更新请求。

（2）事务：InnoDB 存储引擎是唯一支持事务的标准 MySQL 存储引擎，这是管理敏感数据（如金融信息和用户注册信息）的必需软件。

（3）自动灾难恢复：与其他存储引擎不同，InnoDB 表能够自动从灾难中恢复。虽然 MyISAM 表能在灾难后修复，但其过程要长得多。

Oracle 的 InnoDB 存储引擎广泛应用于基于 MySQL 的 Web、电子商务、金融系统、健康护理以及零售应用。因为 InnoDB 可提供高效的 ACID 独立性（Atomicity）、一致性（Consistency）、隔离性（Isolation）、持久性（Durability）兼容事务处理能力，以及独特的高性能和具有可扩展性的构架要素。

另外，InnoDB 设计用于事务处理应用，这些应用需要处理崩溃恢复、参照完整性、高级别的用户并发数，以及响应时间超时服务水平。在 MySQL 5.5 中，最显著的增强性能是将 InnoDB 作为默认的存储引擎。在 MyISAM 以及其他表类型依然可用的情况下，用户无须更改配置，就可构建基于 InnoDB 的应用程序。

5.1.4 MyISAM 存储引擎

MyISAM 存储引擎是 MySQL 中常见的存储引擎，曾是 MySQL 的默认存储引擎。MyISAM 存储引擎是基于 ISAM 存储引擎发展起来的，它解决了 ISAM 的很多不足。MyISAM 增加了很多有用的扩展。

1. MyISAM 存储引擎的文件类型

MyISAM 存储引擎的表存储成 3 个文件。文件的名字与表名相同，扩展名包括 frm、MYD 和 MYI。

（1）frm：存储表的结构。

（2）MYD：存储数据，是 MYData 的缩写。

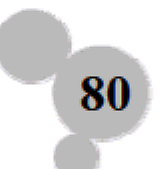
（3）MYI：存储索引，是 MYIndex 的缩写。

2. MyISAM 存储引擎的存储格式

基于 MyISAM 存储引擎的表支持 3 种不同的存储格式，包括静态型、动态型和压缩型。

1）MyISAM 静态

如果所有表列的大小都是静态的（即不使用 xBLOB、xTEXT 或 VARCHAR 数据类型），MySQL



就会自动使用静态 MyISAM 格式。使用这种类型的表性能非常高，因为在维护和访问以预定义格式存储的数据时需要很低的开销。但是，这项优点要以空间为代价，因为每列都需要分配给该列最大空间，而无论该空间是否真正地使用。

2）MyISAM 动态

如果有表列（即使只有一列）定义为动态的（使用 xBLOB、xTEXT 或 VARCHAR），MySQL 就会自动使用动态格式。虽然 MyISAM 动态表占用的空间比静态格式所占空间少，但空间的节省带来了性能的下降。如果某个字段的内容发生改变，则其位置很可能就需要移动，这会导致碎片的产生。随着数据集中的碎片增加，数据访问性能就会相应降低。这个问题有以下两种修复方法。

（1）尽可能使用静态数据类型。

（2）经常使用 OPTIMIZE TABLE 语句，它会整理表的碎片，恢复由于表更新和删除而导致的空间丢失。

3）MyISAM 压缩

有时会创建在整个应用程序生命周期中都只读的表。如果是这种情况，就可以使用 myisampack 工具将其转换为 MyISAM 压缩表来减少空间。在给定硬件配置下（如快速的处理器和低速的硬盘驱动器），性能的提升将相当显著。

3. MyISAM 存储引擎的优缺点

MyISAM 存储引擎的优势在于占用空间小，处理速度快；缺点是不支持事务的完整性和并发性。

5.1.5　MEMORY 存储引擎

MEMORY 存储引擎是 MySQL 中的一类特殊的存储引擎。其使用存储在内存中的内容来创建表，而且所有数据也放在内存中。这些特性都与 InnoDB 存储引擎、MyISAM 存储引擎不同。下面将对 MEMORY 存储引擎的文件存储形式、索引类型、存储周期和优缺点等进行讲解。

1. MEMORY 存储引擎的文件存储形式

每个基于 MEMORY 存储引擎的表实际对应一个磁盘文件。该文件的文件名与表名相同，类型为 frm。该文件中只存储表的结构，而其数据文件都是存储在内存中。这样有利于对数据的快速处理，提高整个表的处理效率。值得注意的是，服务器需要有足够的内存来维持 MEMORY 存储引擎的表的使用。如果不需要使用了，可以释放这些内容，甚至可以删除不需要的表。

2. MEMORY 存储引擎的索引类型

MEMORY 存储引擎默认使用哈希（HASH）索引，其速度要比使用 B 树（BTREE）索引快。如果读者希望使用 B 树索引，可以在创建索引时选择使用。

3. MEMORY 存储引擎的存储周期

MEMORY 存储引擎通常很少用到。因为 MEMORY 表的所有数据是存储在内存上的，如果内存出现异常就会影响到数据的完整性。如果重启机器或者关机，表中的所有数据将消失。因此，基于

MEMORY 存储引擎的表生命周期很短，一般都是一次性的。

4．MEMORY 存储引擎的优缺点

MEMORY 表的大小是受到限制的。表的大小主要取决于两个参数，分别是 max_rows 和 max_heap_table_size。其中，max_rows 可以在创建表时指定；max_heap_table_size 的大小默认为 16MB，可以按需要进行扩大。因此，其存在于内存中的特性，决定了这类表的处理速度非常快。但是，其数据易丢失，生命周期短。

创建 MySQL MEMORY 存储引擎的出发点是速度。为得到最快的响应时间，采用的逻辑存储介质是系统内存。虽然在内存中存储表数据确实会提高性能，但要记住，当 mysqld 守护进程崩溃时，所有的 MEMORY 数据都会丢失。

MEMORY 表不支持 VARCHAR、BLOB 和 TEXT 数据类型，因为这种表类型按固定长度的记录格式存储。此外，如果使用版本 4.1.0 之前的 MySQL，则不支持自动增加列（通过 AUTO_INCREMENT 属性）。当然，要记住 MEMORY 表只用于特殊的范围，不会用于长期存储数据。基于其这个缺陷，选择 MEMORY 存储引擎时要特别小心。

当数据有如下情况时，可以考虑使用 MEMORY 表。

（1）暂时：目标数据只是临时需要，在其生命周期中必须立即可用。

（2）相对无关：存储在 MEMORY 表中的数据如果突然丢失，不会对应用服务产生实质的负面影响，而且不会对数据完整性有长期影响。

如果使用 MySQL 4.1 及其之前版本，MEMORY 的搜索比 MyISAM 表的搜索效率要低，因为 MEMORY 表只支持散列索引，这需要使用整个键进行搜索。但是，4.1 之后的版本同时支持散列索引和 B 树索引。B 树索引优于散列索引的是，可以使用部分查询和通配查询，也可以使用<、>和>=等操作符方便数据挖掘。

5.1.6　如何选择存储引擎

每种存储引擎都有各自的优势，不能笼统地说谁比谁更好，只有适合不适合。下面根据其不同的特性，给出选择存储引擎的建议。

（1）InnoDB 存储引擎：用于事务处理应用程序，具有众多特性，包括 ACID 事务支持，支持外键。同时支持崩溃修复能力和并发控制。如果需要对事务的完整性要求比较高，要求实现并发控制，那选择 InnoDB 存储引擎有其很大的优势。如果需要频繁地进行更新、删除操作的数据库，也可以选择 InnoDB 存储引擎，因为该类存储引擎可以实现事务的提交（Commit）和回滚（Rollback）。

（2）MyISAM 存储引擎：管理非事务表，它提供高速存储和检索，以及全文搜索能力。MyISAM 存储引擎插入数据快，空间和内存使用比较低。如果表主要是用于插入新记录和读出记录，那么选择 MyISAM 存储引擎能实现处理的高效率。如果应用的完整性、并发性要求很低，也可以选择 MyISAM 存储引擎。

（3）MEMORY 存储引擎：MEMORY 存储引擎提供“内存中”的表，MEMORY 存储引擎的所有数据都在内存中，数据的处理速度快，但安全性不高。如果需要很快的读写速度，对数据的安全性要求较低，可以选择 MEMORY 存储引擎。MEMORY 存储引擎对表的大小有要求，不能建太大的表。

所以，这类数据库只使用相对较小的数据库表。

以上存储引擎的选择建议是根据不同存储引擎的特点提出的，并不是绝对的。实际应用中还需要根据各自的实际情况进行分析。

5.1.7　设置数据表的存储引擎

下面创建 db_database03 数据库文件，在数据库中创建 3 个数据表，并分别为其设置不同的存储引擎。以此来诠释这 3 种不同存储引擎创建的数据表文件有什么区别。

（1）创建 tb_001 数据表，设置存储引擎为 MyISAM，生成的数据表文件如图 5.4 所示，由 3 个不同后缀的文件组成。

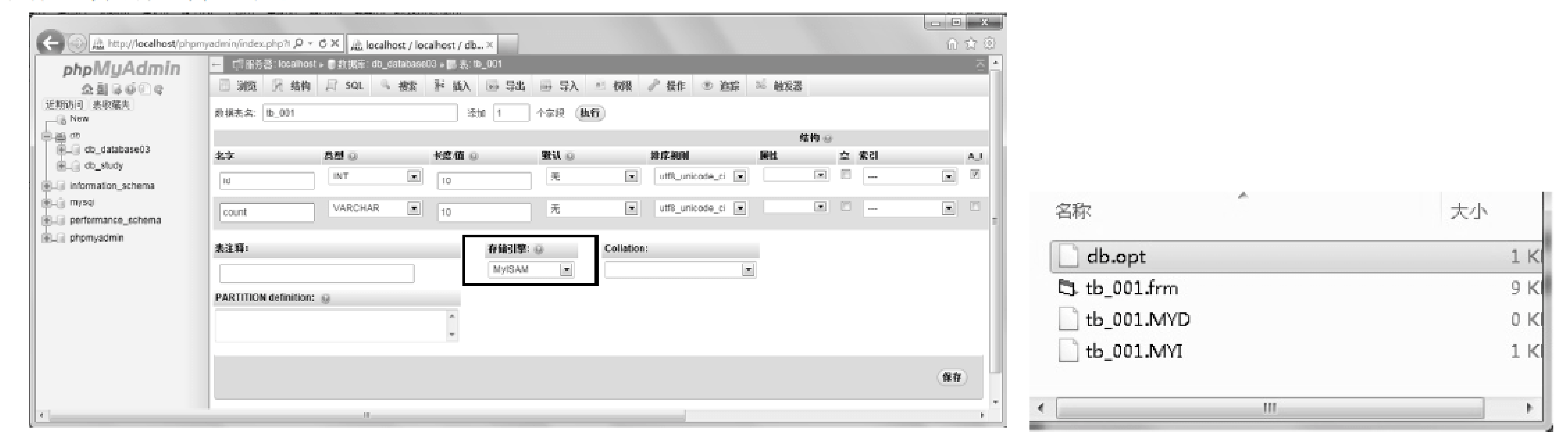


图 5.4　创建 tb_001 数据表及生成的数据表文件

（2）创建 tb_002 数据表，设置存储引擎为 MEMORY，生成的数据表文件如图 5.5 所示，只有一个后缀为 frm 的文件。

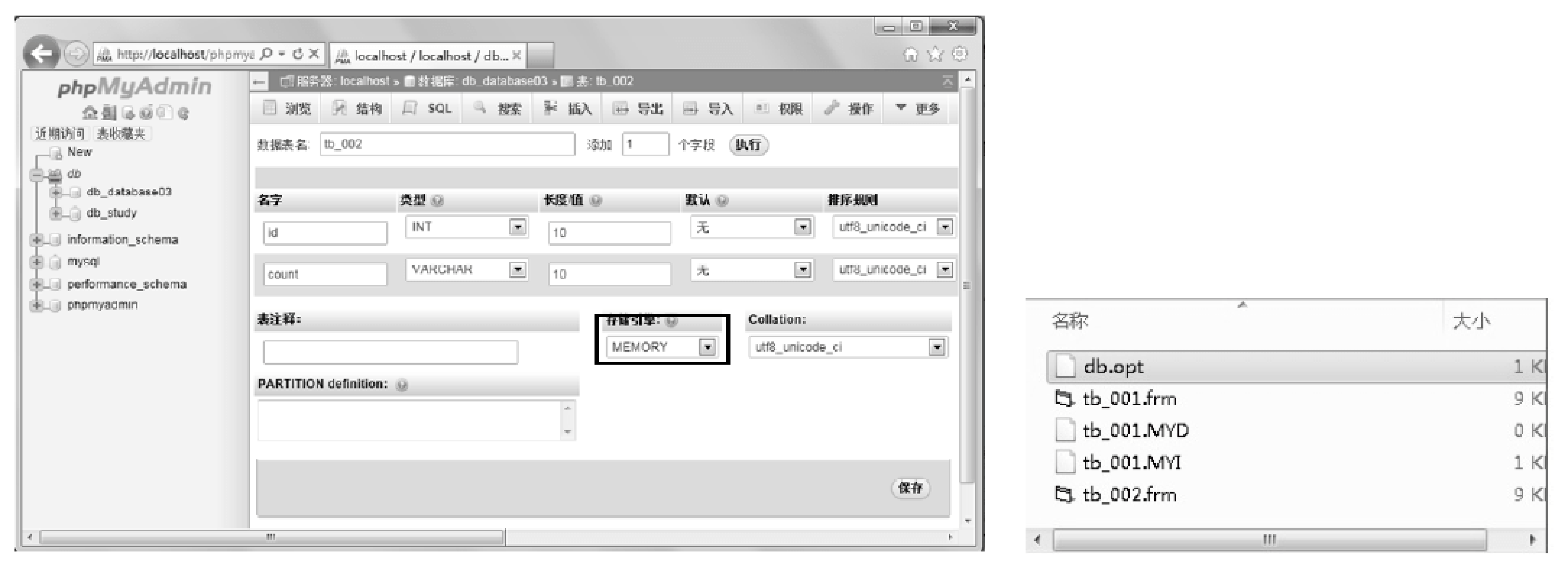


图 5.5　创建 tb_002 数据表及生成的数据表文件

（3）创建 tb_003 数据表，设置存储引擎为 InnoDB，生成的数据表文件如图 5.6 所示，同样也由一个后缀为 frm 的文件组成。

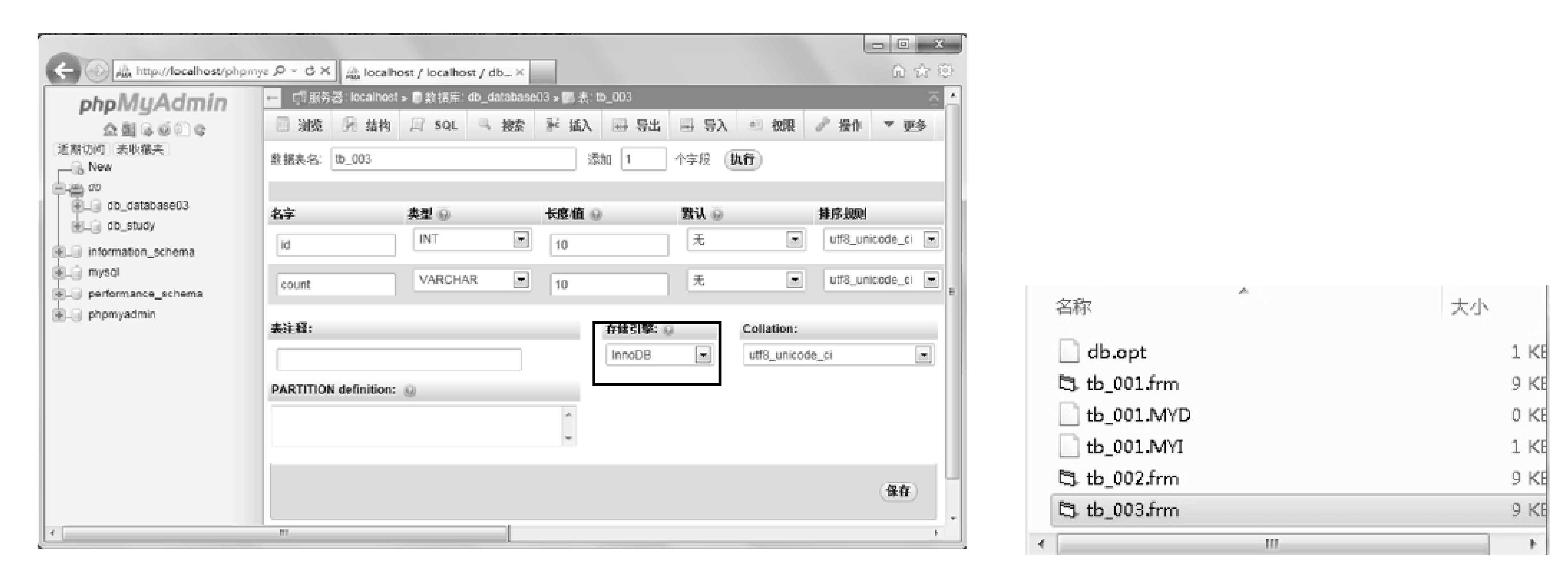


图 5.6　创建 tb_003 数据表及生成的数据表文件

5.2　MySQL 数据类型

在 MySQL 数据库中，每一条数据都有其数据类型。MySQL 支持的数据类型主要分成 3 类：数字类型、字符串（字符）类型、日期和时间类型。

5.2.1　数字类型

MySQL 支持所有的 ANSI/ISO SQL 92 数字类型。这些类型包括准确数字的数据类型（NUMERIC、DECIMAL、INTEGER 和 SMALLINT），还包括近似数字的数据类型（FLOAT、REAL 和 DOUBLE PRECISION）。其中的关键词 INT 是 INTEGER 的同义词，关键词 DEC 是 DECIMAL 的同义词。

数字类型总体可以分成整型和浮点型两类，详细内容如表 5.1 和表 5.2 所示。

表 5.1　整数数据类型

数 据 类 型	取 值 范 围	说　明	单　位
TINYINT	符号值：-127~127　无符号值：0~255	最小的整数	1 字节
BIT	符号值：-127~127　无符号值：0~255	最小的整数	1 字节
BOOL	符号值：-127~127　无符号值：0~255	最小的整数	1 字节
SMALLINT	符号值：-32 768~32 767 无符号值：0~65 535	小型整数	2 字节
MEDIUMINT	符号值：-8 388 608~8 388 607 无符号值：0~16 777 215	中型整数	3 字节
INT	符号值：- 2 147 683 648~2 147 683 647 无符号值：0~4 294 967 295	标准整数	4 字节
BIGINT	符号值： – 9 223 372 036 854 775 808~9 223 372 036 854 775 807 无符号值：0~18 446 744 073 709 551 615	大整数	8 字节

表 5.2　浮点数据类型

数 据 类 型	取 值 范 围	说　明	单　位
FLOAT	+（-）3.402 823 466E+38	单精度浮点数	8 或 4 字节
DOUBLE	+（-）1.797 693 134 862 315 7E+308 +（-）2.225 073 858 507 201 4E-308	双精度浮点数	8 字节
DECIMAL	可变	一般整数	自定义长度


说明

在创建表时，使用哪种数字类型，应遵循以下原则。

（1）选择最小的可用类型，如果值永远不超过 127，则使用 TINYINT 比 INT 强。

（2）对于完全都是数字的，可以选择整数类型。

（3）浮点类型用于可能具有小数部分的数，如货物单价、网上购物交付金额等。

5.2.2　字符串类型

字符串类型可以分为 3 类：普通的文本字符串类型（CHAR 和 VARCHAR）、可变类型（TEXT 和 BLOB）和特殊类型（SET 和 ENUM）。它们之间都有一定的区别，取值范围不同，应用的地方也不同。

（1）普通的文本字符串类型，即 CHAR 和 VARCHAR 类型，CHAR 列的长度被固定为创建表所声明的长度，取值在 1~255 之间；VARCHAR 列的值是变长的字符串，取值和 CHAR 一样。普通的文本字符串类型的介绍如表 5.3 所示。

表 5.3　常规字符串类型

类　型	取 值 范 围	说　明
[national] char(M) [binary|ASCII|unicode]	0~255 个字符	固定长度为 M 的字符串，其中 M 的取值范围为 0~255。national 关键字指定了应该使用的默认字符集。binary 关键字指定了数据是否区分大小写（默认是区分大小写的）。ASCII 关键字指定了在该列中使用 latin1 字符集。unicode 关键字指定了使用 UCS 字符集
char	0~255 个字符	char(M)类似
[national] varchar(M) [binary]	0~255 个字符	长度可变，其他和 char(M)类似

（2）可变类型（TEXT 和 BLOB）。它们的大小可以改变，TEXT 类型适合存储长文本，而 BLOB 类型适合存储二进制数据，支持任何数据，如文本、声音和图像等。TEXT 和 BLOB 类型的介绍如表 5.4 所示。

表 5.4　TEXT 和 BLOB 类型

类　型	最大长度（字节数）	说　明
TINYBLOB	2^8−1（255）	小 BLOB 字段
TINYTEXT	2^8−1（255）	小 TEXT 字段
BLOB	2^16−1（65 535）	常规 BLOB 字段
TEXT	2^16−1（65 535）	常规 TEXT 字段
MEDIUMBLOB	2^24−1（16 777 215）	中型 BLOB 字段
MEDIUMTEXT	2^24−1（16 777 215）	中型 TEXT 字段
LONGBLOB	2^32−1（4 294 967 295）	长 BLOB 字段
LONGTEXT	2^32−1（4 294 967 295）	长 TEXT 字段

（3）特殊类型（SET 和 ENUM）。

特殊类型（SET 和 ENUM）的介绍如表 5.5 所示。

表 5.5　ENUM 和 SET 类型

类　型	最 大 值	说　明
Enum ("value1", "value2", …)	65 535	该类型的列只可以容纳所列值之一或为 NULL
Set ("value1", "value2", …)	64	该类型的列可以容纳一组值或为 NULL

说明

创建表时，使用字符串类型时应遵循以下原则。

（1）从速度方面考虑，要选择固定的列，可以使用 CHAR 类型。

（2）要节省空间，使用动态的列，可以使用 VARCHAR 类型。

（3）要将列中的内容限制在一种选择，可以使用 ENUM 类型。

（4）允许在一个列中有多于一个的条目，可以使用 SET 类型。

（5）如果要搜索的内容不区分大小写，可以使用 TEXT 类型。

（6）如果要搜索的内容区分大小写，可以使用 BLOB 类型。

5.2.3　日期和时间类型

日期和时间类型包括：DATETIME、DATE、TIMESTAMP、TIME 和 YEAR。其中的每种类型都有其取值的范围，如赋予它一个不合法的值，将会被“0”代替。日期和时间类型的介绍如表 5.6 所示。

表 5.6　日期和时间数据类型

类　型	取 值 范 围	说　明
DATE	1000-01-01　9999-12-31	日期，格式 YYYY-MM-DD
TIME	-838:58:59　835:59:59	时间，格式 HH:MM:SS
DATETIME	1000-01-01 00:00:00 9999-12-31 23:59:59	日期和时间，格式 YYYY-MM-DD HH:MM:SS
TIMESTAMP	1970-01-01 00:00:00 2037 年的某个时间	时间标签，在处理报告时使用显示格式取决于 M 的值
YEAR	1901-2155	年份可指定两位数字和四位数字的格式

在 MySQL 中，日期的顺序是按照标准的 ANSI SQL 格式进行输出的。

5.3　小　结

本章对 MySQL 存储引擎和数据类型分别进行了详细讲解，并通过举例说明，帮助读者更好地理解所学知识的用法。在阅读本章时，读者应该重点掌握什么类型的表适合什么类型的存储引擎，同时对 MySQL 中的数据类型也要有一定的了解，在以后设计数据表时，能够合理地选择所使用的数据类型。

5.4　实践与练习

1．查询 MySQL 中支持的存储引擎，并且以友好效果进行显示。**（答案位置：光盘\TM\sl\5\5.2）**
2．查询默认的存储引擎，并且以友好效果进行显示。**（答案位置：光盘\TM\sl\5\5.3）**

第 6 章

操作数据表

（视频讲解：12 分钟）

在对 MySQL 数据表进行操作之前，必须首先使用 USE 语句选择数据库，才可在指定的数据库中对数据表进行操作，如创建数据表、修改表结构、数据表更名或删除数据表等；否则是无法对数据表进行操作的。本章将对数据表的操作方法进行详细介绍。

通过阅读本章，读者可以：

- 掌握创建数据表的方法
- 了解查看数据表的方法
- 掌握修改数据表结构的方法
- 掌握重命名、复制和删除数据表的方法

6.1　创建数据表

创建数据表使用 CREATE TABLE 语句。语法如下。

```
CREATE [TEMPORARY] TABLE [IF NOT EXISTS] 数据表名
[(create_definition,…)][table_options] [select_statement]
```

CREATE TABLE 语句的参数说明如表 6.1 所示。

表 6.1　CREATE TABLE 语句的参数说明

关　键　字	说　　明
TEMPORARY	如果使用该关键字，表示创建一个临时表
IF NOT EXISTS	该关键字用于避免表存在时 MySQL 报告的错误
create_definition	这是表的列属性部分。MySQL 要求在创建表时，表要至少包含一列
table_options	表的一些特性参数，其中大多数选项涉及的是表数据如何存储及存储在何处，如 ENGINE 选项用于定义表的存储引擎。多数情况下，用户不必指定表选项
select_statement	SELECT 语句描述部分，用它可以快速创建表

下面介绍列属性 create_definition 部分，每一列定义的具体格式如下。

```
col_name    type [NOT NULL | NULL] [DEFAULT default_value] [AUTO_INCREMENT]
            [PRIMARY KEY ] [reference_definition]
```

属性 create_definition 的参数说明如表 6.2 所示。

表 6.2　属性 create_definition 的参数说明

参　　数	说　　明
col_name	字段名
type	字段类型
NOT NULL | NULL	指出该列是否允许是空值，系统一般默认允许为空值，所以当不允许为空值时，必须使用 NOT NULL
DEFAULT default_value	表示默认值
AUTO_INCREMENT	表示是否是自动编号，每个表只能有一个 AUTO_INCREMENT 列，并且必须被索引
PRIMARY KEY	表示是否为主键。一个表只能有一个 PRIMARY KEY。如表中没有一个 PRIMARY KEY，而某些应用程序需要 PRIMARY KEY，MySQL 将返回第一个没有任何 NULL 列的 UNIQUE 键，作为 PRIMARY KEY
reference_definition	为字段添加注释

以上是创建一个数据表的一些基础知识，它看起来十分复杂，但在实际的应用中使用最基本的格式创建数据表即可，具体格式如下。

```
CREATE TABLE 数据表名 (列名 1 属性,列名 2 属性…);
```

例 6.1　使用 CREATE TABLE 语句在 MySQL 数据库 db_admin 中创建一个名为 tb_admin 的数据表，该表包括 id、user、password 和 createtime 等字段，具体代码如下。（**实例位置：光盘\TM\sl\6\6.1**）

```
USE db_admin;
CREATE TABLE tb_admin(
    id int auto_increment primary key,
    user varchar(30) not null,
    password varchar(30) not null,
    createtime datetime);
```

执行结果如图 6.1 所示。

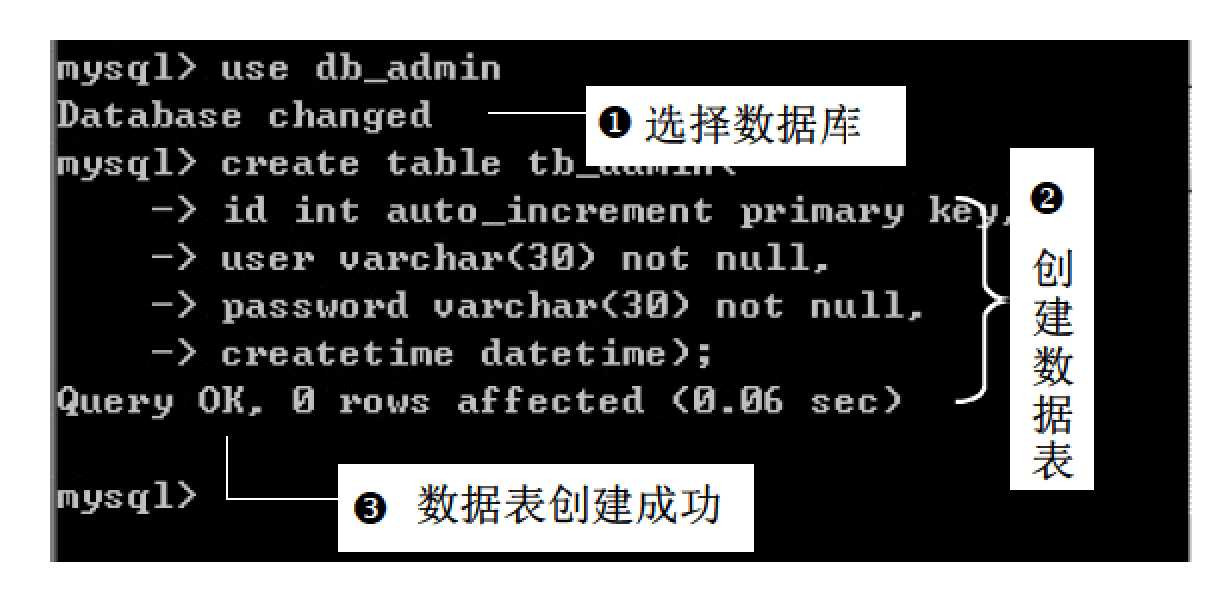


图 6.1　创建 MySQL 数据表



说明

在完成本实例前，如果不存在名称为 db_admin 的数据库，那么需要先创建该数据库，创建数据库 db_admin 的具体代码如下。

```
CREATE DATABASE db_admin;
```

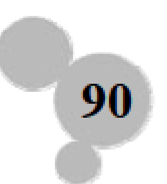
6.2　查看表结构

对于一个创建成功的数据表，可以使用 SHOW COLUMNS 语句或 DESCRIBE 语句查看指定数据表的表结构。下面分别对这两个语句进行介绍。

6.2.1　使用 SHOW COLUMNS 语句查看

在 MySQL 中，使用 SHOW COLUMNS 语句可以查看表结构，SHOW COLUMNS 语句的基本语法格式如下。

```
SHOW   [FULL] COLUMNS   FROM 数据表名 [FROM 数据库名];
```



或

```
SHOW  [FULL] COLUMNS  FROM 数据表名.数据库名;
```

例 6.2　使用 SHOW COLUMNS 语句查看数据表 tb_admin 的表结构，具体代码如下。（**实例位置：光盘\TM\sl\6\6.2**）

```
SHOW COLUMNS FROM tb_admin FROM db_admin;
```

执行效果如图 6.2 所示。

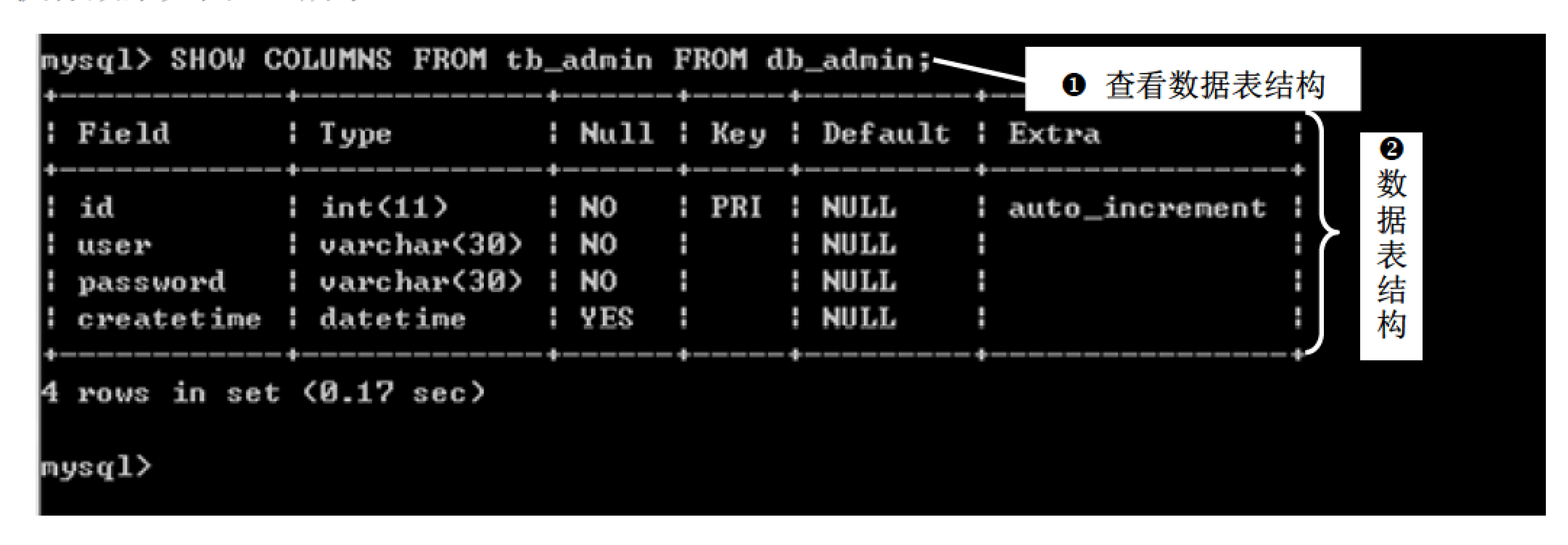


图 6.2　查看表结构

6.2.2　使用 DESCRIBE 语句查看

在 MySQL 中，还可以使用 DESCRIBE 语句查看数据表结构。DESCRIBE 语句的基本语法格式如下。

```
DESCRIBE 数据表名;
```

其中，DESCRIBE 可以简写成 DESC。在查看表结构时，也可以只列出某一列的信息。其语法格式如下。

```
DESCRIBE 数据表名 列名;
```

例 6.3　使用 DESCRIBE 语句的简写形式查看数据表 tb_admin 中的某一列信息。（**实例位置：光盘\TM\sl\6\6.3**）

（1）编写 SQL 语句，选择要查看数据表所在的数据库，具体代码如下。

```
USE db_admin;
```

（2）应用简写的 DESC 命令查看数据表 tb_admin 中的 user 字段的信息，具体代码如下。

```
DESC tb_admin user;
```

执行结果如图 6.3 所示。

```
mysql> USE db_admin;
Database changed
mysql> DESC tb_admin user;
+-------+-------------+------+-----+---------+-------+
| Field | Type        | Null | Key | Default | Extra |
+-------+-------------+------+-----+---------+-------+
| user  | varchar(30) | NO   |     | NULL    |       |
+-------+-------------+------+-----+---------+-------+
1 row in set (0.02 sec)

mysql>
```

❶ 选择数据表所在的数据库

❷ 使用简写的 DESC 查看数据表某一列的信息

❸ 数据表结构

图 6.3　查看表的某一列信息

6.3　修改表结构

修改表结构使用 ALTER TABLE 语句。修改表结构指增加或者删除字段、修改字段名称或者字段类型、设置取消主键外键、设置取消索引以及修改表的注释等，语法如下。

```
ALTER [IGNORE] TABLE 数据表名 alter_spec[,alter_spec]...| table_options
```

参数说明如下。

（1）[IGNORE]：可选项，表示如果出现重复关键的行，则只执行一行，其他重复的行被删除。

（2）数据表名：用于指定要修改的数据表的名称。

（3）alter_spec 子句：用于定义要修改的内容，其语法格式如下。

```
ADD [COLUMN] create_definition [FIRST | AFTER column_name ]              //添加新字段
 | ADD INDEX [index_name] (index_col_name,...)                           //添加索引名称
 | ADD PRIMARY KEY (index_col_name,...)                                  //添加主键名称
 | ADD UNIQUE [index_name] (index_col_name,...)                          //添加唯一索引
 | ALTER [COLUMN] col_name {SET DEFAULT literal | DROP DEFAULT}          //修改字段默认值
 | CHANGE [COLUMN] old_col_name create_definition                        //修改字段名/类型
 | MODIFY [COLUMN] create_definition                                     //修改子句定义字段
 | DROP [COLUMN] col_name                                                //删除字段名称
 | DROP PRIMARY KEY                                                      //删除主键名称
 | DROP INDEX index_name                                                 //删除索引名称
 | RENAME [AS] new_tbl_name                                              //更改表名
```

上面的语法中，各参数说明如下。

① create_definition：用于定义列的数据类型和属性，与 6.1 节 CREATE TABLE 语句中的语法相同。

② [FIRST | AFTER column_name]：用于指定位于哪个字段的前面或者后面，当使用 FIRST 关键字时，表示位于指定字段的前面；使用 AFTER 关键字时，表示位于指定字段的后面。其中的 column_name 表示字段名。

③ [index_name]：可选项，用于指定索引名。

④ (index_col_name,...)：用于指定索引列名。



⑤ {SET DEFAULT literal | DROP DEFAULT}子句：为字段设置或者删除默认值。其中 literal 参数为要设置的默认值。

⑥ old_col_name：用于指定要修改的字段名。

⑦ new_tbl_name：用于指定新的表名。

（4）table_options：用于指定表的一些特性参数，其中大多数选项涉及的是表数据如何存储及存储在何处，如 ENGINE 选项用于定义表的存储引擎。多数情况下，用户不必指定表选项。

说明

ALTER TABLE 语句允许指定多个动作，其动作间使用逗号分隔，每个动作表示对表的一个修改。

6.3.1　添加新字段及修改字段定义

在 MySQL 的 ALTER TABLE 语句中，可以通过使用 ADD [COLUMN] create_definition [FIRST | AFTER column_name]子句来添加新字段；使用 MODIFY [COLUMN] create_definition 子句可以修改已定义字段的定义。下面将通过一个具体实例演示如何为一个已有表添加新字段，并修改已有字段的字段定义。

例 6.4　添加一个新的字段 email，类型为 varchar(50)，not null，将字段 user 的类型由 varchar(30)改为 varchar(40)。**（实例位置：光盘\TM\sl\6\6.4）**

（1）选择数据库 db_admin，具体代码如下。

```
USE db_admin;
```

（2）编写 SQL 语句，实现向数据表 tb_admin 中添加一个新字段，并且修改字段 user 的类型，具体代码如下。

```
ALTER TABLE tb_admin ADD email varchar(50) not null ,
modify user varchar(40);
```

在命令行模式下的运行情况如图 6.4 所示。

```
mysql> USE db_admin;
Database changed
mysql> ALTER TABLE tb_admin ADD email varchar(50) not null ,
    -> modify user varchar(40);
Query OK, 1 row affected (0.05 sec)
Records: 1  Duplicates: 0  Warnings: 0

mysql> _
```

❶ 选择数据表所在的数据库

❷ 添加新字段并修改字段 user 的字段类型

图 6.4　添加新字段、修改字段类型

（3）通过 DESC 命令查看数据 tb_user 的表结构，以查看表结构是否成功修改，具体代码如下。

```
DESC tb_admin;
```

执行效果如图 6.5 所示。

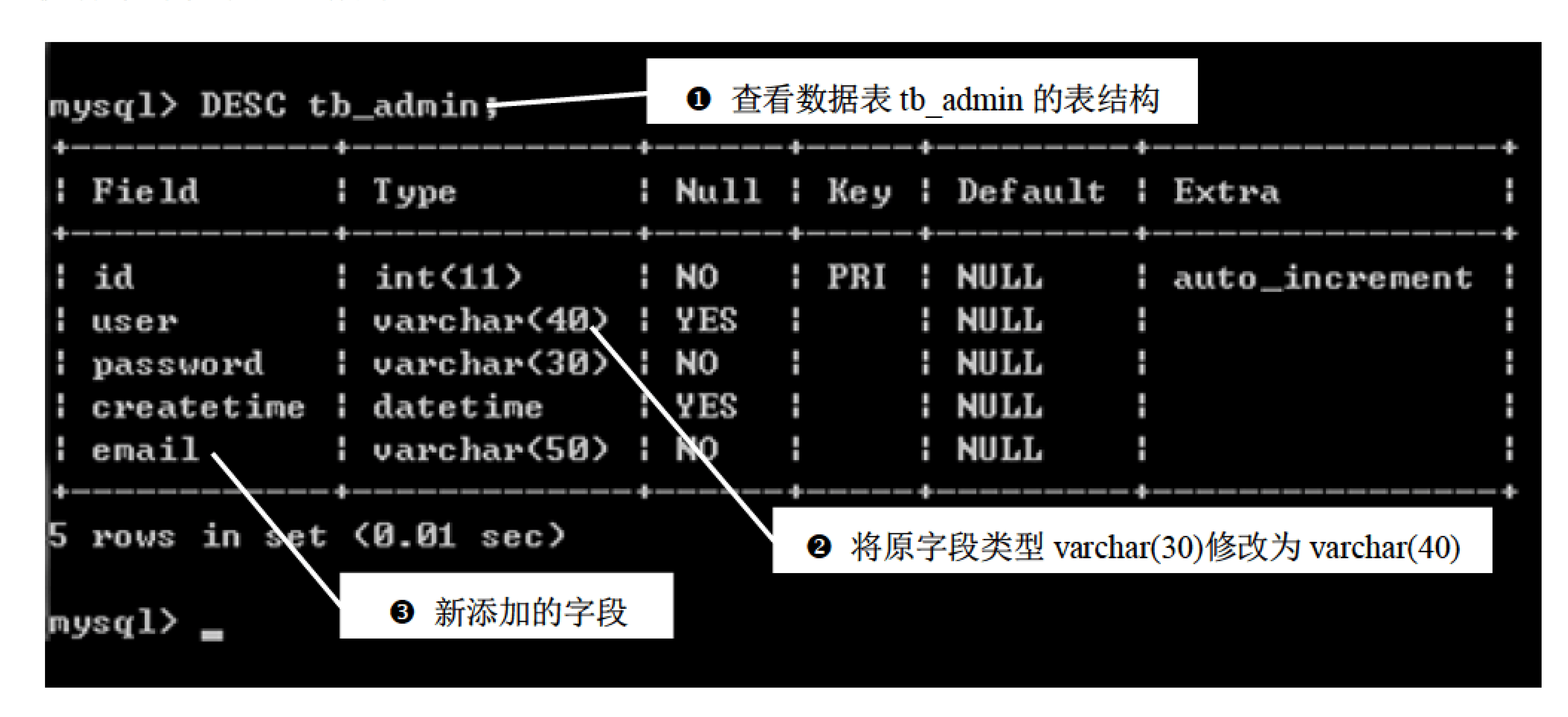


图 6.5　修改后 tb_user 的表结构



说明

通过 ALTER 语句修改表列，其前提是必须将表中数据全部删除，然后才可以修改表列。

6.3.2　修改字段名

在 MySQL 的 ALTER TABLE 语句中，使用 CHANGE [COLUMN] old_col_name create_definition 子句可以修改字段名或者字段类型。下面将通过一个具体实例演示如何修改字段名。

例 6.5　将数据表 tb_userNew1 的字段名 user 修改为 username，具体代码如下。**（实例位置：光盘\TM\sl\6\6.5）**

```
ALTER TABLE db_admin.tb_usernew1
CHANGE COLUMN user username VARCHAR(30) NULL DEFAULT NULL ;
```

执行效果如图 6.6 所示。

```
mysql> ALTER TABLE db_admin.tb_usernew1
    -> CHANGE COLUMN user username VARCHAR(30) NULL DEFAULT NULL ;
Query OK, 0 rows affected (0.30 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql>
```

图 6.6　修改字段名



6.3.3　删除字段

在 MySQL 的 ALTER TABLE 中，使用 DROP [COLUMN] col_name 子句可以删除指定字段。下面将通过一个具体实例演示如何删除字段。

例 6.6　将数据库 db_admin 中的数据表 tb_userNew1 更名为 tb_userOld。**（实例位置：光盘\TM\sl\6\6.6）**

（1）选择数据库 db_admin，具体代码如下。

```
USE db_admin;
```

（2）编写 SQL 语句，实现将数据表 tb_admin 中的字段 email 删除，具体代码如下。

```
ALTER TABLE tb_admin DROP email;
```

在命令行模式下的运行情况如图 6.7 所示。

```
mysql> USE db_admin;
Database changed
mysql> ALTER TABLE tb_admin DROP email;
Query OK, 1 row affected (0.06 sec)
Records: 1  Duplicates: 0  Warnings: 0

mysql> _
```

图 6.7　删除字段

6.3.4　修改表名

在 MySQL 的 ALTER TABLE 中，使用 RENAME [AS] new_tbl_name 子句可以修改表名。下面将通过一个具体实例演示如何修改表名。

例 6.7　将数据库 db_admin 中的数据表 tb_userNew1 更名为 tb_userOld。**（实例位置：光盘\TM\sl\6\6.7）**

（1）选择数据库 db_admin，具体代码如下。

```
USE db_admin;
```

（2）编写 SQL 语句，实现将数据表 tb_userNew1 更名为 tb_userOld，具体代码如下。

```
ALTER TABLE tb_usernew1 RENAME AS tb_userOld;
```

在命令行模式下的运行情况如图 6.8 所示。

```
mysql> USE db_admin;
Database changed
mysql> ALTER TABLE tb_usernew1 RENAME AS tb_userOld;
Query OK, 0 rows affected (0.26 sec)

mysql>
```

图 6.8　修改表名

6.4 重 命 名 表

在 MySQL 中，重命名数据表可以使用 RENAME TABLE 语句来实现。RENAME TABLE 语句的基本语法格式如下。

```
RENAME TABLE 数据表名 1 To 数据表名 2
```



说明

该语句可以同时对多个数据表进行重命名，多个表之间以逗号“,”分隔。

例 6.8　对数据表 tb_admin 进行重命名，更名后的数据表为 tb_user。**（实例位置：光盘\TM\sl\6\6.8）**

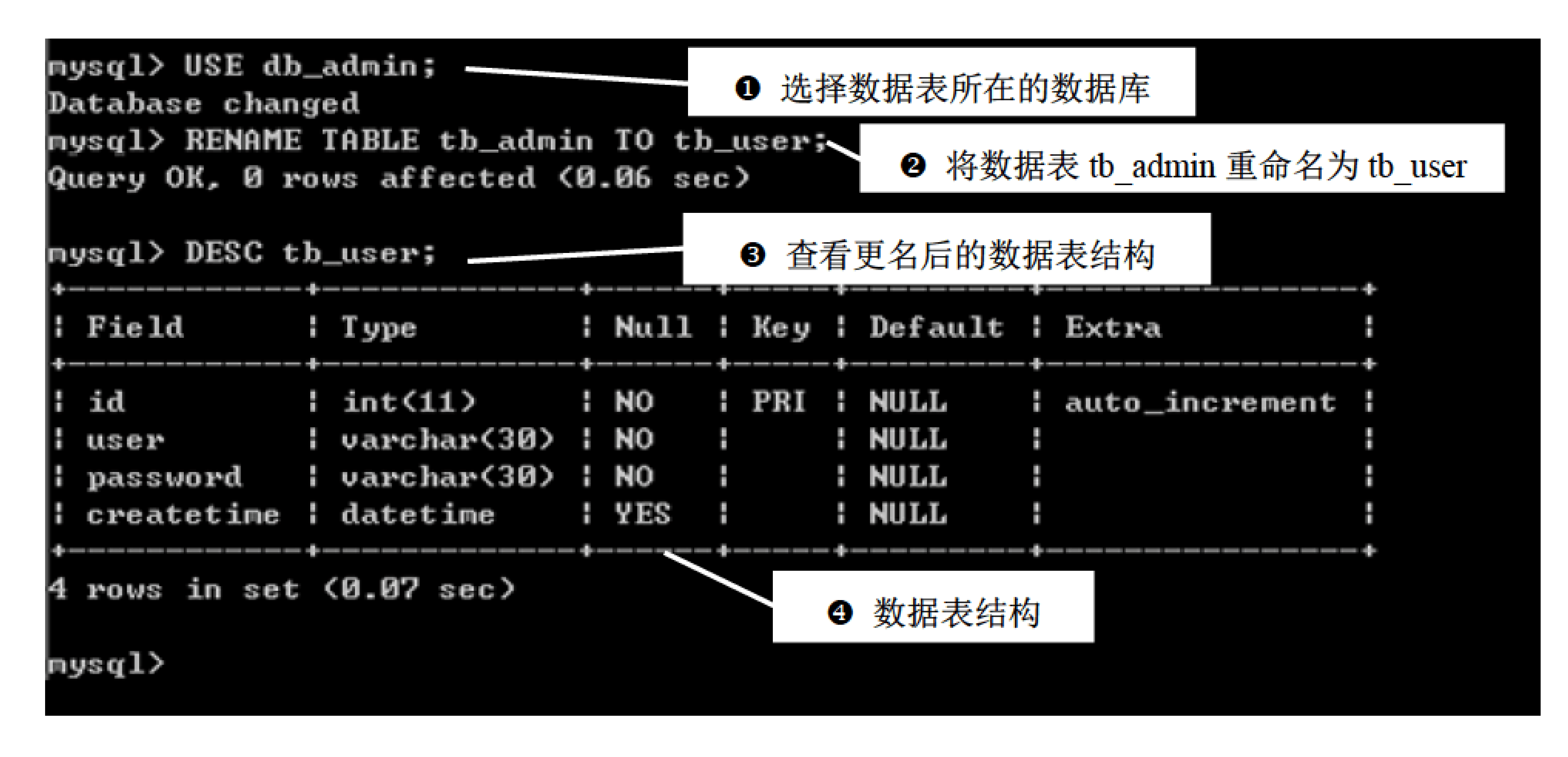
（1）使用 RENAME 语句将数据表 tb_admin 重命名为 tb_user，具体代码如下。

```
RENAME TABLE tb_admin TO tb_user;
```

（2）重命名后，应用 DESC 语句查看数据表 tb_user 的表结构，具体代码如下。

```
DESC tb_user;
```

执行效果如图 6.9 所示。



```
mysql> USE db_admin;
Database changed
mysql> RENAME TABLE tb_admin TO tb_user;
Query OK, 0 rows affected (0.06 sec)

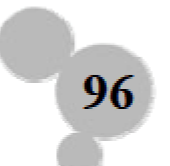
mysql> DESC tb_user;
+------------+-------------+------+-----+---------+----------------+
| Field      | Type        | Null | Key | Default | Extra          |
+------------+-------------+------+-----+---------+----------------+
| id         | int(11)     | NO   | PRI | NULL    | auto_increment |
| user       | varchar(30) | NO   |     | NULL    |                |
| password   | varchar(30) | NO   |     | NULL    |                |
| createtime | datetime    | YES  |     | NULL    |                |
+------------+-------------+------+-----+---------+----------------+
4 rows in set (0.07 sec)

mysql>
```


图 6.9　对数据表进行更名

6.5 复 制 表

创建表的 CREATE TABLE 命令还有另外一种语法结构，在一张已经存在的数据表的基础上创建



一份该表的备份，也就是复制表。这种用法的语法格式如下。

```
CREATE TABLE [IF NOT EXISTS] 数据表名
    {LIKE 源数据表名 | (LIKE 源数据表名)}
```

参数说明如下。

（1）[IF NOT EXISTS]：可选项，如果使用该子句，表示当要创建的数据表名不存在时，才会创建。如果不使用该子句，当要创建的数据表名存在时，将出现错误。

（2）数据表名：表示新创建的数据表的名，该数据表名必须是在当前数据库中不存在的表名。

（3）{LIKE 源数据表名 | (LIKE 源数据表名)}：必选项，用于指定依照哪个数据表来创建新表，也就是要为哪个数据表创建副本。

说明

使用该语法复制数据表时，将创建一个与源数据表相同结构的新表，该数据表的列名、数据类型空指定和索引都将被复制，但是表的内容是不会复制的。因此，新创建的表是一张空表。如果想要复制表中的内容，可以通过使用 AS(查询表达式)子句来实现。

例 6.9 在数据库 db_admin 中创建一份数据表 tb_user 的备份 tb_userNew。**（实例位置：光盘\TM\sl\6\6.9）**

（1）选择数据表所在的数据库 db_admin，具体代码如下。

```
USE db_admin;
```

（2）创建一份数据表 tb_user 的备份 tb_userNew，具体代码如下。

```
CREATE TABLE tb_userNew
    LIKE tb_user;
```

执行效果如图 6.10 所示。

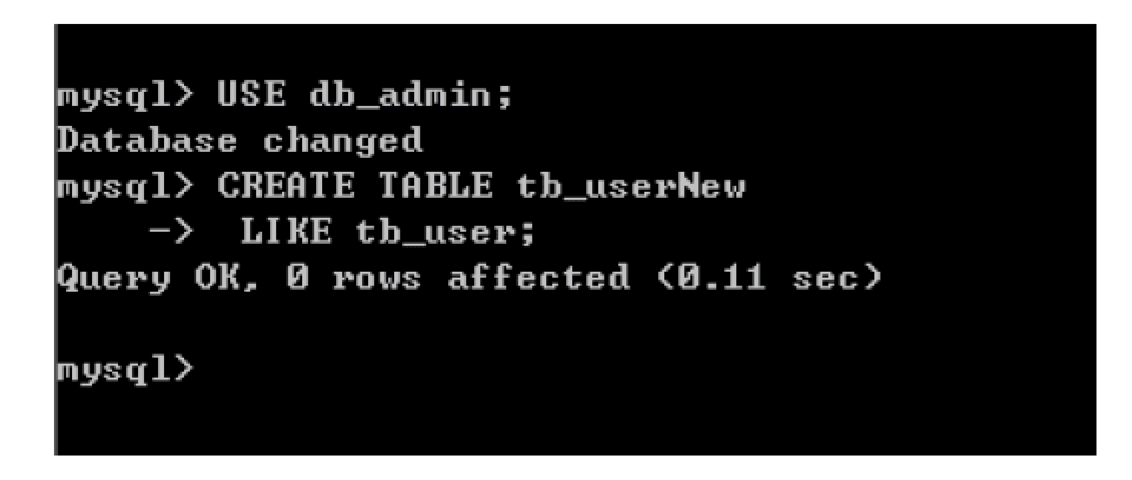
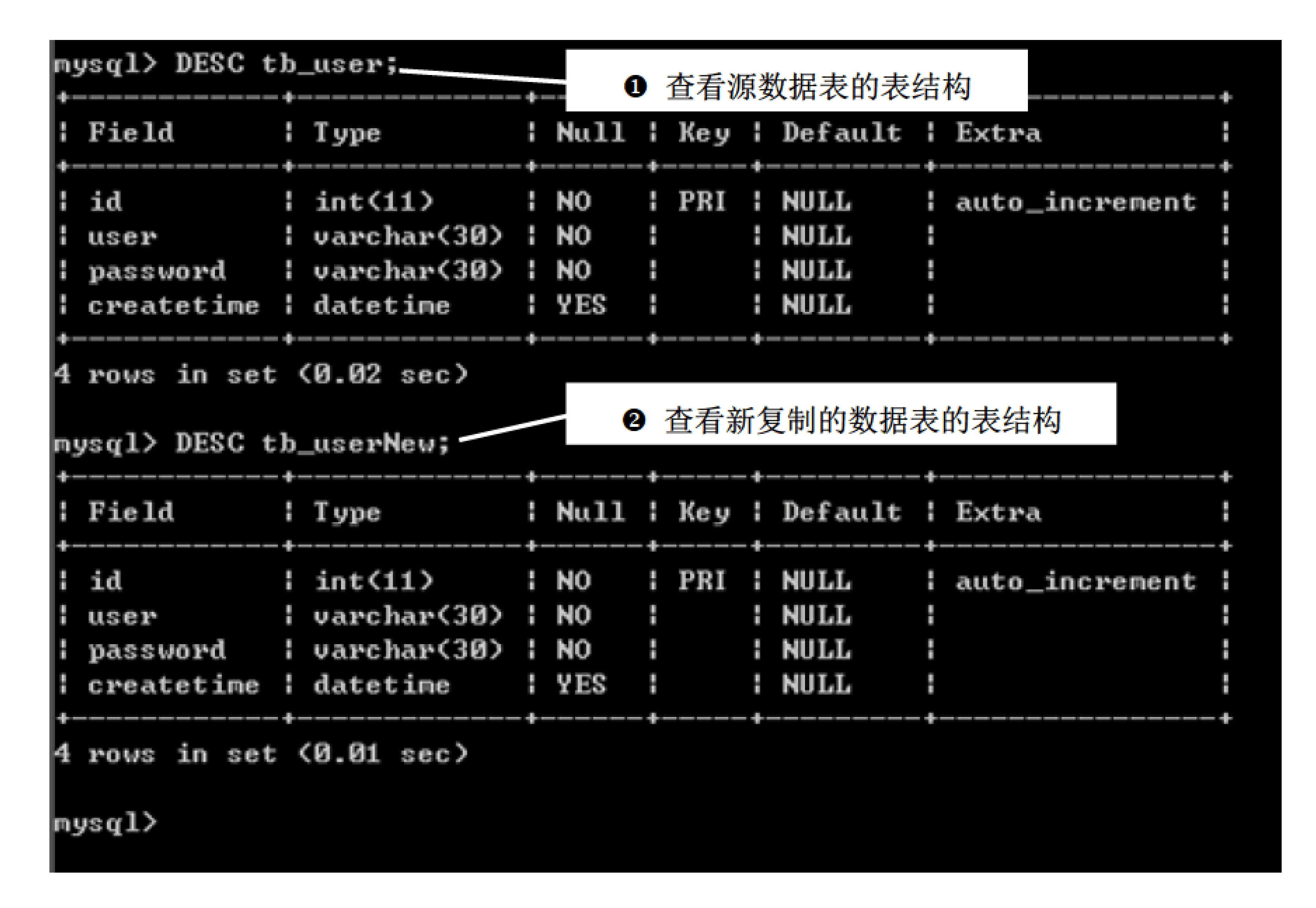


图 6.10　创建一份数据表 tb_user 的备份 tb_userNew

（3）查看数据表 tb_user 和 tb_userNew 的表结构，具体代码如下。

```
DESC tb_user;
DESC tb_userNew;
```

执行结果如图 6.11 所示。




```
mysql> DESC tb_user;
+------------+-------------+------+-----+---------+----------------+
| Field      | Type        | Null | Key | Default | Extra          |
+------------+-------------+------+-----+---------+----------------+
| id         | int(11)     | NO   | PRI | NULL    | auto_increment |
| user       | varchar(30) | NO   |     | NULL    |                |
| password   | varchar(30) | NO   |     | NULL    |                |
| createtime | datetime    | YES  |     | NULL    |                |
+------------+-------------+------+-----+---------+----------------+
4 rows in set (0.02 sec)

mysql> DESC tb_userNew;
+------------+-------------+------+-----+---------+----------------+
| Field      | Type        | Null | Key | Default | Extra          |
+------------+-------------+------+-----+---------+----------------+
| id         | int(11)     | NO   | PRI | NULL    | auto_increment |
| user       | varchar(30) | NO   |     | NULL    |                |
| password   | varchar(30) | NO   |     | NULL    |                |
| createtime | datetime    | YES  |     | NULL    |                |
+------------+-------------+------+-----+---------+----------------+
4 rows in set (0.01 sec)

mysql>
```

图 6.11　查看数据表 tb_user 和 tb_userNew 的表结构

从图 6.11 中可以看出，数据表 tb_user 和 tb_userNew 的表结构是一样的。

（4）分别查看数据表 tb_user 和 tb_userNew 的内容，具体代码如下。

```
SELECT * FROM tb_user;
SELECT * FROM tb_userNew;
```

执行效果如图 6.12 所示。

```
mysql> SELECT * FROM tb_user;
+----+------+----------+---------------------+
| id | user | password | createtime          |
+----+------+----------+---------------------+
|  1 | mr   | 111      | 2014-09-11 10:12:13 |
+----+------+----------+---------------------+
1 row in set (0.00 sec)

mysql> SELECT * FROM tb_userNew;
Empty set (0.00 sec)

mysql>
```

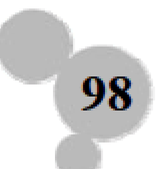
图 6.12　查看数据表 tb_user 和 tb_userNew 的内容

从图 6.12 中可以看出，在复制表时并没有复制表中的数据。

（5）如果在复制数据表时，想要同时复制其中的内容，那么需要使用下面的代码来实现。

```
CREATE TABLE tb_userNew1
      AS SELECT * FROM tb_user;
```

执行结果如图 6.13 所示。



```
mysql> CREATE TABLE tb_userNew1
    -> AS SELECT * FROM tb_user;
Query OK, 1 row affected (0.35 sec)
Records: 1  Duplicates: 0  Warnings: 0

mysql> _
```

图 6.13　复制数据表同时复制其中的数据

（6）查看数据表 tb_userNew1 中的数据，具体代码如下。

```
SELECT * FROM tb_userNew1;
```

执行效果如图 6.14 所示。

```
mysql> SELECT * FROM tb_userNew1;
+----+------+----------+---------------------+
| id | user | password | createtime          |
+----+------+----------+---------------------+
|  1 | mr   | 111      | 2014-09-11 10:12:13 |
+----+------+----------+---------------------+
1 row in set (0.02 sec)

mysql> _
```

图 6.14　查看新复制的数据表 tb_userNew1 的数据

从图 6.14 中可以看出，在复制表的同时也复制了表中的数据。

6.6　删　除　表

删除数据表的操作很简单，同删除数据库的操作类似，使用 DROP TABLE 语句即可实现。DROP TABLE 语句的基本语法格式如下。

```
DROP TABLE [IF EXISTS] 数据表名;
```

参数说明如下。

（1）[IF EXISTS]：可选项，用于在删除表前先判断是否存在要删除的表，只有存在时，才执行删除操作，这样可以避免要删除的表不存在时出现错误信息。

（2）数据表名：用于指定要删除的数据表名，可以同时删除多张数据表，多个数据表名之间用英文半角的逗号“,”分隔。

例 6.10　删除数据表 tb_user。**（实例位置：光盘\TM\sl\6\6.10）**

（1）选择数据表所在的数据库 db_admin，具体代码如下。

```
USE db_admin;
```

（2）应用 DROP TABLE 语句删除数据表 tb_user，具体代码如下。

```
DROP TABLE;
```

执行效果如图 6.15 所示。



图 6.15　删除数据表

注意

删除数据表的操作应该谨慎使用。一旦删除了数据表，那么表中的数据将会全部清除，没有备份则无法恢复。

在删除数据表的过程中，删除一个不存在的表将会产生错误，如果在删除语句中加入 IF EXISTS 关键字就不会出错了，格式如下。

```
DROP TABLE IF EXISTS 数据表名;
```

6.7　小　　结

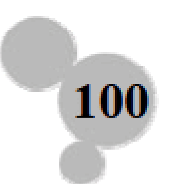
本章主要介绍了如何创建数据表、查看表结构、修改表结构、重命名表、复制表和删除表等内容。其中，创建和修改表这两部分内容比较重要，需要不断的练习才会对这两部分了解得更加透彻。而且，这两部分很容易出现语法错误，必须在练习中掌握正确的语法规则。创建表和修改表后一定要查看表的结构，这样可以确认操作是否正确。删除表时一定要特别小心，因为删除表的同时会删除表中的所有数据。

6.8　实践与练习

1．编写 SQL 语句，实现查看数据表 tb_admin 的表结构。**（答案位置：光盘\TM\sl\6\6.11）**

2．编写 SQL 语句，实现为数据表 tb_userNew 的 user 字段设置默认值为 mr。**（答案位置：光盘\TM\sl\6\6.12）**

3．编写 SQL 语句，实现当数据表 tb_user 存在的情况下删除该数据表。**（答案位置：光盘\TM\sl\6\6.13）**



第2篇

核心技术

本篇介绍MySQL基础，表数据的增、删、改操作，数据查询，常用函数、索引、视图的使用等。学习完这一部分，读者能够了解和熟悉MySQL及其常用函数，使用SQL操作MySQL数据库中的视图，掌握SQL查询、子查询、嵌套查询、连接查询的用法等。

第 7 章

MySQL 基础

（视频讲解：24 分钟）

同其他语言一样，MySQL 数据库也有自己的运算符和流程控制语句。本章将对 MySQL 的运算符和流程控制语句进行详细介绍。

通过阅读本章，读者可以：

- 掌握 MySQL 的运算符
- 掌握 MySQL 的流程控制语句

7.1 运　算　符

7.1.1　算术运算符

算术运算符是 MySQL 中最常用的一类运算符。MySQL 支持的算术运算符包括加、减、乘、除、求余。如表 7.1 所示为算术运算符的符号、作用、表达式的形式。

表 7.1　算术运算符

符　　号	作　　用
+	加法运算
-	减法运算
*	乘法运算
/	除法运算
%	求余运算
DIV	除法运算，返回商。同“/”
MOD	求余运算，返回余数。同“%”

说明

加（+）、减（-）和乘（*）可以同时运算多个操作数。除号（/）和求余运算符（%）也可以同时计算多个操作数，但是这两个符号计算多个操作数不太好。DIV 和 MOD 这两个运算符只有两个参数。进行除法和求余的运算时，如果 x2 参数是 0 时，计算结果将是空值（NULL）。

例 7.1　使用算术运算符对数据表 tb_book1 中的 row 字段值进行加、减、乘、除运算，计算结果如图 7.1 所示。**（实例位置：光盘\TM\sl\7\7.1）**

```
nysql> select row,row+row,row-row,row*row,row/row from tb_book1;
+------+---------+---------+---------+---------+
| row  | row+row | row-row | row*row | row/row |
+------+---------+---------+---------+---------+
| 12   |      24 |       0 |     144 |       1 |
| 95   |     190 |       0 |    9025 |       1 |
| NULL |       0 |       0 |       0 |    NULL |
| 1    |       2 |       0 |       1 |       1 |
| 8    |      16 |       0 |      64 |       1 |
| NULL |       0 |       0 |       0 |    NULL |
+------+---------+---------+---------+---------+
5 rows in set, 16 warnings (0.00 sec)
```

图 7.1　使用算术运算符计算数据

结果输出了 row 字段的原值，以及执行算术运算符后得到的值。

7.1.2 比较运算符

比较运算符是查询数据时最常用的一类运算符。SELECT 语句中的条件语句经常要使用比较运算符。通过这些比较运算符，可以判断表中的哪些记录是符合条件的。比较运算符的符号、名称和应用示例如表 7.2 所示。

表 7.2　比较运算符

运 算 符	名　称	示　例	运 算 符	名　称	示　例
=	等于	id=5	IS NOT NULL	n/a	id IS NOT NULL
>	大于	id>5	BETWEEN AND	n/a	id BETWEEN1 AND 15
<	小于	id<5	IN	n/a	id IN (3,4,5)
>=	大于等于	id=>5	NOT IN	n/a	name NOT IN (shi,li)
<=	小于等于	id<=5	LIKE	模式匹配	name LIKE ('shi%')
!=或<>	不等于	id!=5	NOT LIKE	模式匹配	name NOT LIKE ('shi%')
IS NULL	n/a	id is null	REGEXP	常规表达式	name 正则表达式

下面对几种较常用的比较运算符进行详解。

1．运算符“=”

“=”用来判断数字、字符串和表达式等是否相等。如果相等，返回 1，否则返回 0。

说明

在运用运算符“=”判断两个字符是否相同时，数据库系统都是根据字符的 ASCII 码进行判断的。如果 ASCII 码相等，则表示这两个字符相同。如果 ASCII 码不相等，则表示两个字符不同。切记空值（NULL）不能使用“=”来判断。

例 7.2　运用运算符“=”查询出 id 等于 27 的记录，查询结果如图 7.2 所示。**（实例位置：光盘\TM\sl\7\7.2）**



图 7.2　使用“=”查询记录



从结果中可以看出，id 等于 27 的记录返回值为 1，id 不等于 27 的记录，返回值则为 0。

2．运算符“<>”和“!=”

“<>”和“!=”用来判断数字、字符串、表达式等是否不相等。如果不相等，则返回 1；否则，返回 0。这两个符号也不能用来判断空值（NULL）。

例 7.3　运用运算符“<>”和“!=”判断数据表 tb_book 中的 row 字段值是否等于 1、41 或 24。运算结果如图 7.3 所示。**（实例位置：光盘\TM\sl\7\7.3）**

```
mysql> select id,books,row<>1,row!=41,row!=24 from tb_book where id=27;
+----+---------------+--------+---------+---------+
| id | books         | row<>1 | row!=41 | row!=24 |
+----+---------------+--------+---------+---------+
| 27 | PHP典型模块        |      1 |       1 |       1 |
+----+---------------+--------+---------+---------+
1 row in set (0.00 sec)

mysql>
```

图 7.3　使用运算符“<>”和“!=”判断数据

结果显示返回值都为 1，这表示记录中的 row 字段值不等于 1、41、24。

3．运算符“>”

“>”用来判断左边的操作数是否大于右边的操作数。如果大于，返回 1；否则，返回 0。同样空值（NULL）不能使用“>”来判断。

例 7.4　使用运算符“>”来判断数据表 tb_book 中的 row 字段值是否大于 90，是则返回 1，否则返回 0，空值返回 NULL。运算结果如图 7.4 所示。**（实例位置：光盘\TM\sl\7\7.4）**

```
mysql> select id,books,row>90 from tb_book;
+----+-----------------------+--------+
| id | books                 | row>90 |
+----+-----------------------+--------+
| 26 | JAVA典型模块                |      0 |
| 27 | PHP典型模块                 |      1 |
| 28 | C#项目整合                  |   NULL |
| 29 | aaaa                  |      0 |
| 30 | aa                    |      0 |
| 25 | JAVA范例完全自学手册            |   NULL |
+----+-----------------------+--------+
6 rows in set (0.00 sec)

mysql>
```

图 7.4　使用运算符“>”查询数据


说明

运算符“<”、运算符“<=”和运算符“>=”都与运算符“>”如出一辙，其使用方法基本相同，这里不再赘述。

4．运算符 IS NULL

IS NULL 用来判断操作数是否为空值（NULL）。操作数为 NULL 时，结果返回 1；否则，返回 0。IS NOT NULL 刚好与 IS NULL 相反。

例 7.5　用运算符 IS NULL 来判断数据表 tb_book 中的 row 字段值是否为空值，查询结果如图 7.5 所示。（**实例位置：光盘\TM\sl\7\7.5**）

```
mysql> select id,books,row IS NULL from tb_book;
+----+---------------------------+-------------+
| id | books                     | row IS NULL |
+----+---------------------------+-------------+
| 26 | JAVA典型模块                |           0 |
| 27 | PHP典型模块                 |           0 |
| 28 | C#项目整合                  |           1 |
| 29 | aaaa                      |           0 |
| 30 | aa                        |           0 |
| 25 | JAVA范例完全自学手册            |           1 |
+----+---------------------------+-------------+
6 rows in set (0.02 sec)

mysql>
```

图 7.5　使用运算符 IS NULL 来判断字段值是否为空

结果显示，row 字段值为空的返回值为 1，不为空的返回值为 0。

说明

"=" "<>" "!=" ">" ">=" "<" "<=" 等运算符都不能用来判断空值（NULL）。一旦使用，结果将返回 NULL。如果要判断一个值是否为空值，可以使用 "<=>"、IS NULL 和 IS NOT NULL 来判断。注意：NULL 和' NULL'是不同的，前者表示为空值，后者表示一个由 4 个字母组成的字符串。

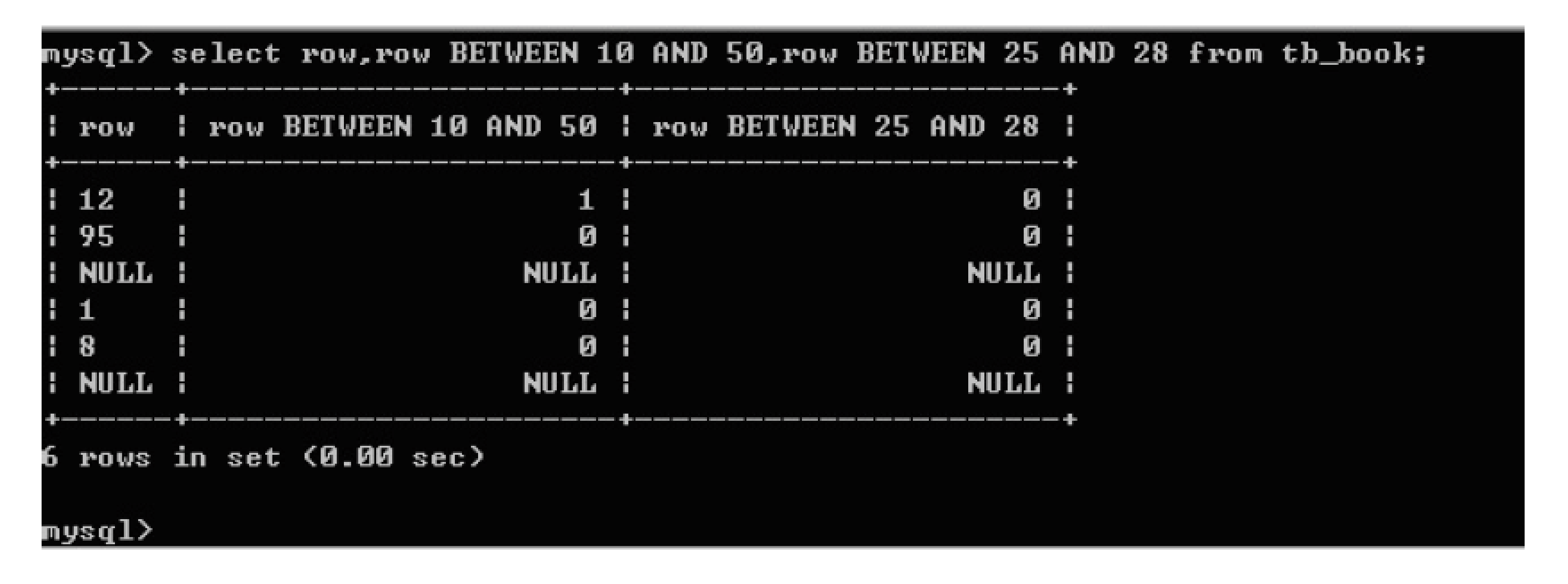
5．运算符 BETWEEN AND

BETWEEN AND 用于判断数据是否在某个取值范围内，其表达式如下。

```
x1 BETWEEN m AND n
```

如果 x1 大于等于 m，且小于等于 n，结果将返回 1，否则将返回 0。

例 7.6　运用运算符 BETWEEN AND 判断数据表 tb_book 中的 row 字段值是否在 10~50 及 25~28 之间，查询结果如图 7.6 所示。（**实例位置：光盘\TM\sl\7\7.6**）

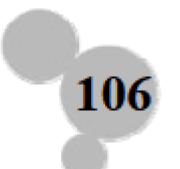


```
mysql> select row,row BETWEEN 10 AND 50,row BETWEEN 25 AND 28 from tb_book;
+------+-----------------------+-----------------------+
| row  | row BETWEEN 10 AND 50 | row BETWEEN 25 AND 28 |
+------+-----------------------+-----------------------+
| 12   |                     1 |                     0 |
| 95   |                     0 |                     0 |
| NULL |                  NULL |                  NULL |
| 1    |                     0 |                     0 |
| 8    |                     0 |                     0 |
| NULL |                  NULL |                  NULL |
+------+-----------------------+-----------------------+
6 rows in set (0.00 sec)

mysql>
```

图 7.6　使用运算符 BETWEEN AND 判断 row 字段值的范围

从查询结果中可以看出，在范围内则返回 1，否则返回 0，空值返回 NULL。



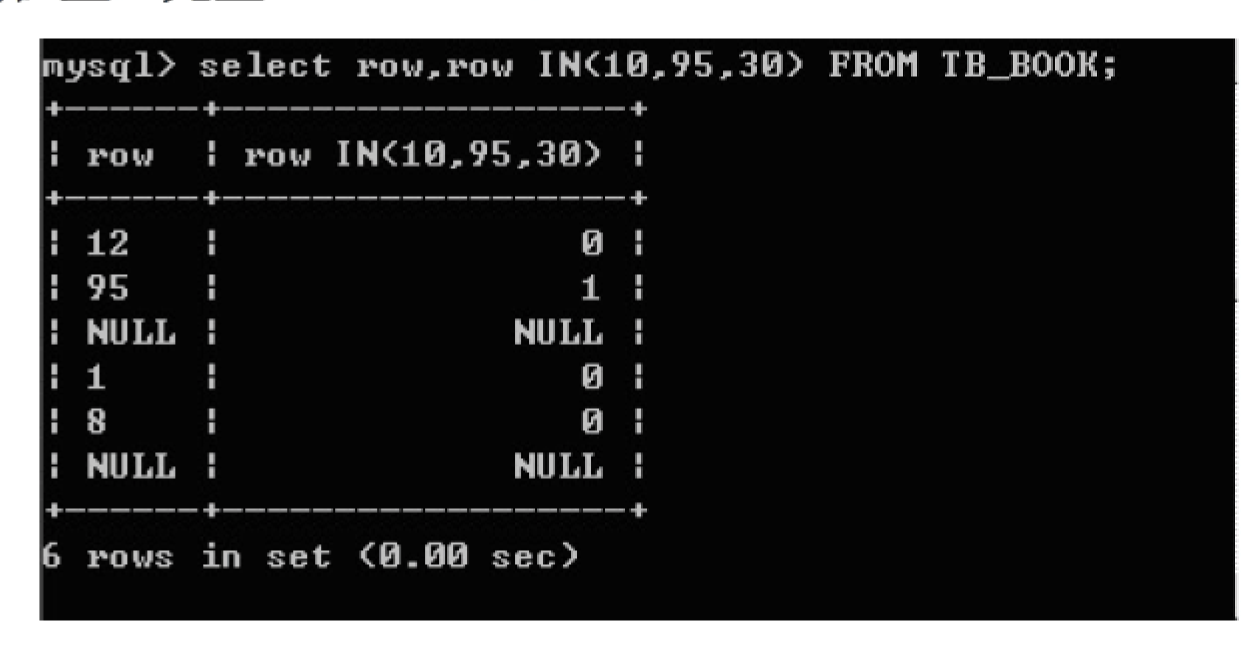
6．运算符 IN

IN 用于判断数据是否存在于某个集合中，其表达式如下。

```
x1 IN(值 1,值 2,⋯值 n)
```

如果 x1 等于值 1 到值 n 中的任何一个值，结果将返回 1；如果不是，结果将返回 0。

例 7.7　下面运用运算符 IN 判断数据表 tb_book 中的 row 字段值是否在指定的范围内，查询结果如图 7.7 所示。（**实例位置：光盘\TM\sl\7\7.7**）



```
mysql> select row,row IN(10,95,30) FROM TB_BOOK;
+------+------------------+
| row  | row IN(10,95,30) |
+------+------------------+
| 12   |                0 |
| 95   |                1 |
| NULL |             NULL |
| 1    |                0 |
| 8    |                0 |
| NULL |             NULL |
+------+------------------+
6 rows in set (0.00 sec)
```

图 7.7　使用运算符 IN 判断 row 字段值的范围

查询结果如图 7.7 所示，在范围内则返回 1，否则返回 0，空值返回 NULL。

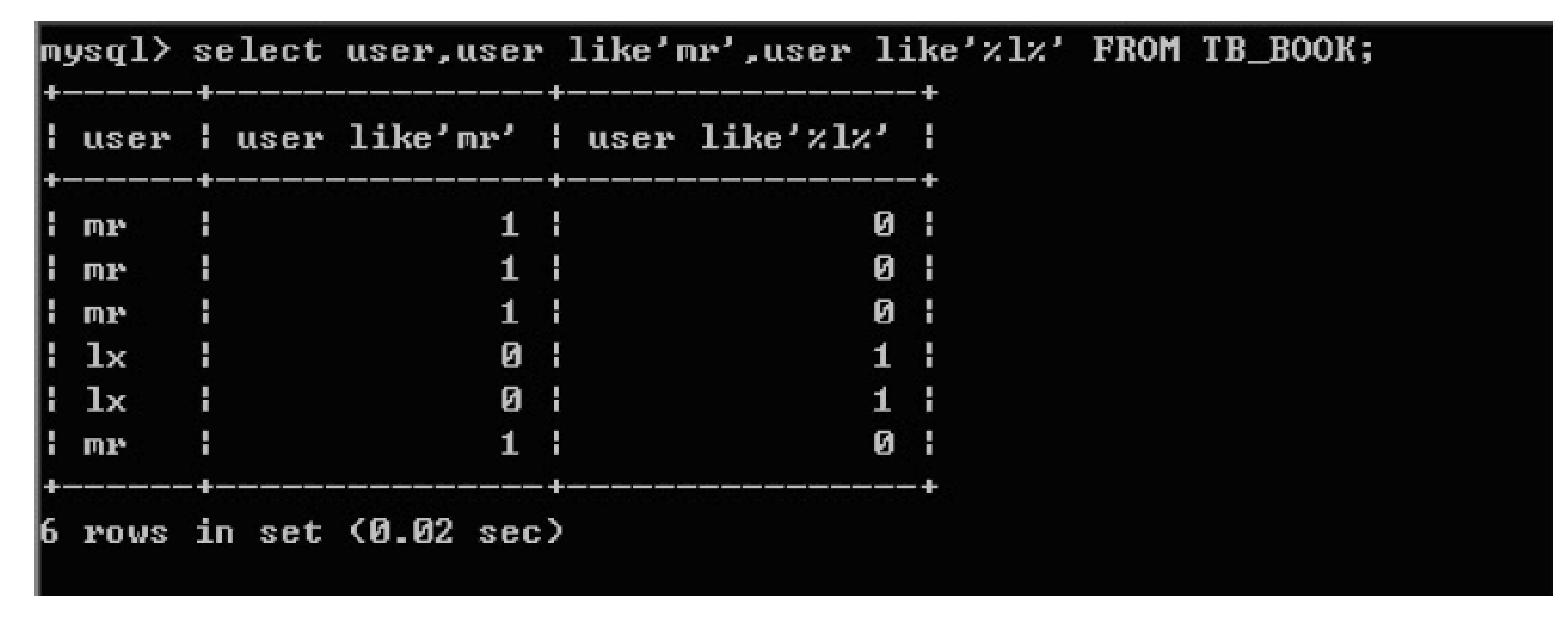
7．运算符 LIKE

LIKE 用来匹配字符串，其表达式如下。

```
x1 LIKE s1
```

如果 x1 与字符串 s1 匹配，结果将返回 1；否则返回 0。

例 7.8　使用运算符 LIKE 判断数据表 tb_book 中的 user 字段值是否与指定的字符串匹配，查询结果如图 7.8 所示。（**实例位置：光盘\TM\sl\7\7.8**）



```
mysql> select user,user like'mr',user like'%l%' FROM TB_BOOK;
+------+---------------+----------------+
| user | user like'mr' | user like'%l%' |
+------+---------------+----------------+
| mr   |             1 |              0 |
| mr   |             1 |              0 |
| mr   |             1 |              0 |
| lx   |             0 |              1 |
| lx   |             0 |              1 |
| mr   |             1 |              0 |
+------+---------------+----------------+
6 rows in set (0.02 sec)
```

图 7.8　使用运算符 LIKE 判断 user 字段是否匹配某字符

查询结果如图 7.8 所示，user 字段值为 mr 字符的记录，结果则返回 1，否则返回 0；user 字段值中包含 l 字符的记录，匹配则返回 1，否则返回 0。

8．运算符 REGEXP

REGEXP 同样用于匹配字符串，但其使用的是正则表达式进行匹配，其表达式如下。

```
x1 REGEXP '匹配方式'
```

如果 x1 满足匹配方式，结果将返回 1；否则将返回 0。

例 7.9 使用运算符 REGEXP 来匹配 user 字段的值是否以指定字符开头、结尾，同时是否包含指定的字符串，执行结果如图 7.9 所示。**（实例位置：光盘\TM\sl\7\7.9）**

```
mysql> select user,user REGEXP'm',user regexp'g$',user regexp'^m' FROM TB_BOOK;
+------+----------------+----------------+----------------+
| user | user REGEXP'm' | user regexp'g$' | user regexp'^m' |
+------+----------------+----------------+----------------+
| mr   |              1 |              0 |              1 |
| mr   |              1 |              0 |              1 |
| mr   |              1 |              0 |              1 |
| lx   |              0 |              0 |              0 |
| lx   |              0 |              0 |              0 |
| mr   |              1 |              0 |              1 |
+------+----------------+----------------+----------------+
6 rows in set (0.00 sec)

mysql> _
```

图 7.9 使用 REGEXP 运算符匹配字符串

本例使用运算符 REGEXP 判断数据表 tb_book 表中的 user 字段值是否以 m 字符开头；是否以 g 字符结尾；在 user 字段值中是否包含 m 字符，如果满足条件则返回 1，否则返回 0。

说明

使用运算符 REGEXP 匹配字符串，其使用方法非常简单。REGEXP 运算符经常与 “^” “$” 和 “.” 一起使用。 “^” 用来匹配字符串的开始部分； “$” 用来匹配字符串的结尾部分； “.” 用来代表字符串中的一个字符。

7.1.3 逻辑运算符

逻辑运算符用来判断表达式的真假。如果表达式是真，结果返回 1；如果表达式是假，结果返回 0。逻辑运算符又称为布尔运算符。MySQL 中支持 4 种逻辑运算符，分别是与、或、非和异或。如表 7.3 所示为 4 种逻辑运算符的符号及作用。

表 7.3 逻辑运算符

符 号	作 用
&&或 AND	与
||或 OR	或
!或 NOT	非
XOR	异或

1. 与运算

“&&”或者 AND 是与运算的两种表达方式。如果所有数据不为 0 且不为空值（NULL）时，结果返回 1；如果存在任何一个数据为 0 时，结果返回 0；如果存在一个数据为 NULL 且没有数据为 0 时，结果返回 NULL。与运算符支持多个数据同时进行运算。

例 7.10　运用运算符“&&”判断 row 字段的值是否存在 0 或者 NULL（“row&&1”（row 字段值与 1）和“row&&0”（row 字段值与 0）），如果存在则返回 1，否则返回 0，空值返回 NULL。执行结果如图 7.10 所示。**（实例位置：光盘\TM\sl\7\7.10）**

```
mysql> select row,row&&1,row&&0 from tb_book;
+------+--------+--------+
| row  | row&&1 | row&&0 |
+------+--------+--------+
| 12   |      1 |      0 |
| 95   |      1 |      0 |
| NULL |   NULL |      0 |
| 1    |      1 |      0 |
| 8    |      1 |      0 |
| NULL |   NULL |      0 |
+------+--------+--------+
6 rows in set (0.00 sec)

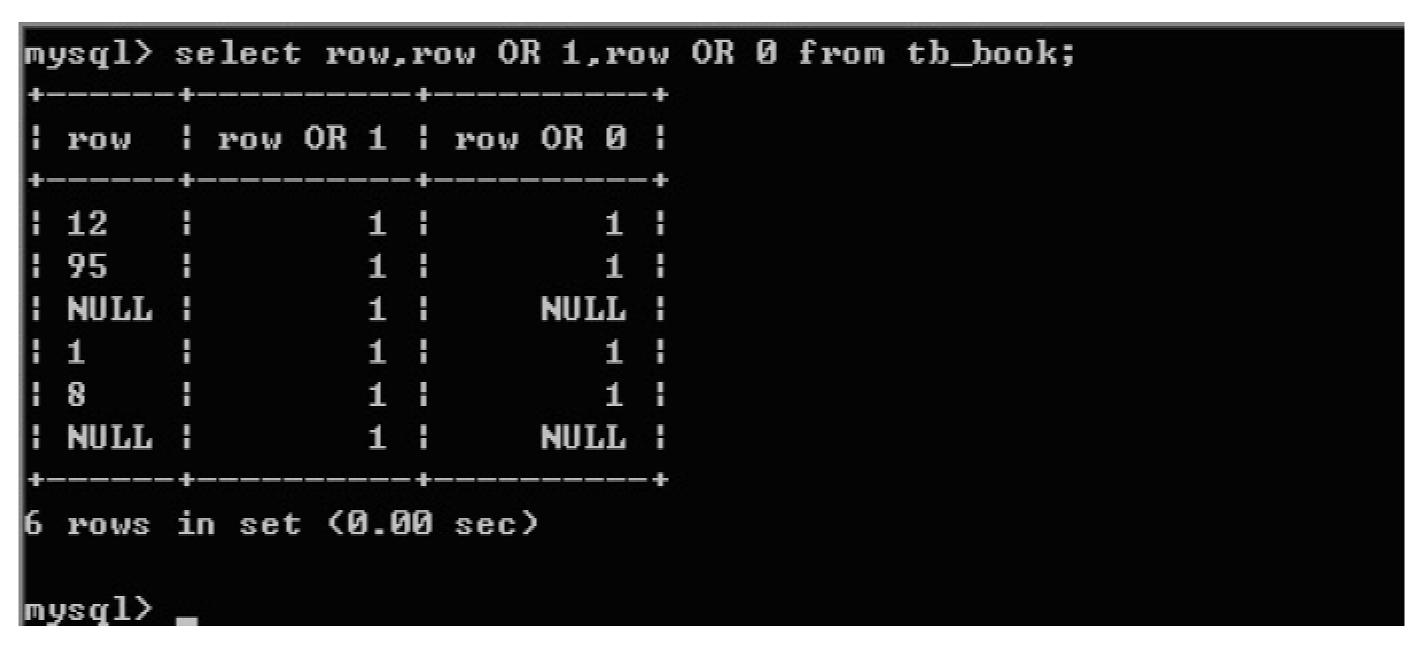
mysql>
```

图 7.10　使用运算符&&判断数据

2. 或运算

“||”或者 OR 表示或运算。所有数据中存在任何一个数据为非 0 的数字时，结果返回 1；如果数据中不包含非 0 的数字，但包含 NULL 时，结果返回 NULL；如果操作数中有 0 时，结果返回 0。或运算符“||”也可以同时操作多个数据。

例 7.11　运用运算符 OR 判断数据表 tb_book 中 row 字段是否包含 NULL 或者非 0 数字(“row OR 1”和“row OR 0”）。执行结果如图 7.11 所示。**（实例位置：光盘\TM\sl\7\7.11）**



```
mysql> select row,row OR 1,row OR 0 from tb_book;
+------+----------+----------+
| row  | row OR 1 | row OR 0 |
+------+----------+----------+
| 12   |        1 |        1 |
| 95   |        1 |        1 |
| NULL |        1 |     NULL |
| 1    |        1 |        1 |
| 8    |        1 |        1 |
| NULL |        1 |     NULL |
+------+----------+----------+
6 rows in set (0.00 sec)

mysql> _
```

图 7.11　使用运算符 OR 匹配数据

结果显示，“row OR 1”中包含 NULL 和 1 这个非 0 的数字，所以返回结果为 1；“row OR 0”中包含非 0 的数字、NULL 和 0 的数字，所以返回 NULL 和 1。

3. 非运算

“!”或者 NOT 表示非运算。通过非运算，将返回与操作数据相反的结果。如果操作数据是非 0

的数字，结果返回 0；如果操作数据是 0，结果返回 1；如果操作数据是 NULL，结果返回 NULL。

例 7.12 运用运算符“!”判断 tb_book 表中 row 字段的值是否为 0 或者 NULL。执行结果如图 7.12 所示。**（实例位置：光盘\TM\sl\7\7.12）**

```
mysql> select row,!row from tb_book;
+------+------+
| row  | !row |
+------+------+
| 12   |    0 |
| 95   |    0 |
| NULL | NULL |
| 1    |    0 |
| 8    |    0 |
| NULL | NULL |
+------+------+
6 rows in set (0.00 sec)

mysql> _
```

图 7.12　使用运算符“!”判断数据

结果显示，row 字段中值为 NULL 的记录，返回值为 NULL，不为 0 的记录，返回值为 0。

4．异或运算

XOR 表示异或运算。只要其中任何一个操作数据为 NULL 时，结果返回 NULL；如果两个操作数都是非 0 值，或者都是 0，则返回结果为 0；如果一个为 0，另一个为非 0 值，返回结果是 1。

例 7.13 使用运算符 XOR 判断数据表 tb_book 中 row 字段值是否为 NULL（“row XOR 1”和“row XOR 0”）。执行结果如图 7.13 所示。**（实例位置：光盘\TM\sl\7\7.13）**

```
mysql> select row,row XOR 1,row XOR 0 from tb_book;
+------+-----------+-----------+
| row  | row XOR 1 | row XOR 0 |
+------+-----------+-----------+
| 12   |         0 |         1 |
| 95   |         0 |         1 |
| NULL |      NULL |      NULL |
| 1    |         0 |         1 |
| 8    |         0 |         1 |
| NULL |      NULL |      NULL |
+------+-----------+-----------+
6 rows in set (0.00 sec)

mysql> _
```

图 7.13　使用运算符 XOR 判断数据

结果显示，“row XOR 1”中 row 字段中的值为非 0 数字和 NULL 值，所以返回值为 0 和 NULL；“row XOR 0”中包含 0，所以返回值为 1；而 row 字段值为 NULL 的记录，返回值则为 NULL。

7.1.4　位运算符

位运算符是在二进制数上进行计算的运算符。位运算会先将操作数变成二进制数再进行位运算，然后再将计算结果从二进制数变回十进制数。MySQL 中支持 6 种位运算符，分别是按位与、按位或、按位取反、按位异或、按位左移和按位右移。6 种位运算符的符号及作用如表 7.4 所示。

表 7.4　位运算符

符　　号	作　　用
&	按位与。进行该运算时，数据库系统会先将十进制数转换为二进制数。然后对应操作数的每个二进制位上进行与运算。1 和 1 相与得 1，与 0 相与得 0。运算完成后再将二进制数变回十进制数
|	按位或。将操作数化为二进制数后，每位都进行或运算。1 和任何数或运算的结果都是 1，0 与 0 或运算的结果为 0
~	按位取反。将操作数化为二进制数后，每位都进行取反运算。1 取反后变成 0，0 取反后变成 1
^	按位异或。将操作数化为二进制数后，每位都进行异或运算。相同的数异或的结果是 0，不同的数异或的结果为 1
<<	按位左移。“m<<n”表示 m 的二进制数向左移 n 位，右边补上 n 个 0。例如，二进制数 001 左移 1 位后将变成 0010
>>	按位右移。“m>>n”表示 m 的二进制数向右移 n 位，左边补上 n 个 0。例如，二进制数 011 右移 1 位后变成 001，最后一个 1 直接被移出

例 7.14　将数字 4 和 6 进行按位与、按位或，并将 4 按位取反。执行结果如图 7.14 所示。**（实例位置：光盘\TM\sl\7\7.14）**

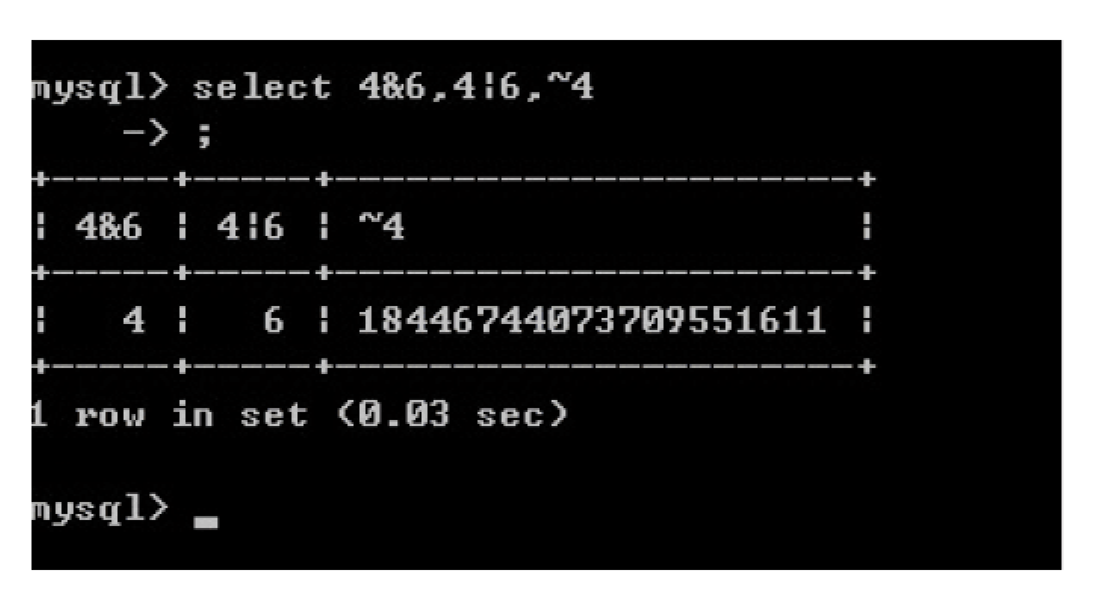


图 7.14　位运算的实例

7.1.5　运算符的优先级

由于在实际应用中可能需要同时使用多个运算符。这就必须考虑运算符的运算顺序。正所谓：闻道有先后，术业有专攻。

本节将具体阐述 MySQL 运算符使用的优先级，如表 7.5 所示。按照从高到低，从左到右的级别进行运算操作。如果优先级相同，则表达式左边的运算符先运算。

表 7.5　MySQL 运算符的优先级

优　先　级	运　算　符
1	!
2	~
3	^

续表

优　先　级	运　算　符
4	*,/,DIV,%,MOD
5	+,-
6	>>,<<
7	&
8	|
9	=,<=>,<,<=,>,>=,!=,<>,IN,IS,NULL,LIKE,REGEXP
10	BETWEEN AND,CASE,WHEN,THEN,ELSE
11	NOT
12	&&,AND
13	||,OR,XOR
14	:=

7.2　流程控制语句

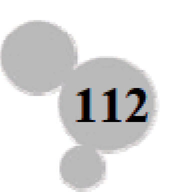
在 MySQL 中，常见的过程式 SQL 语句可以用在一个存储过程体中。其中包括 IF 语句、CASE 语句、LOOP 语句、WHILE 语句、ITERATE 语句和 LEAVE 语句，它们可以进行流程控制。

7.2.1　IF 语句

IF 语句用来进行条件判断，根据不同的条件执行不同的操作。该语句在执行时首先判断 IF 后的条件是否为真，则执行 THEN 后的语句，如果为假则继续判断 IF 语句直到为真为止，当以上都不满足时则执行 ELSE 语句后的内容。IF 语句表示形式如下。

```
IF condition THEN
        …
[ELSE condition THEN]
  …
[ELSE]
…
ENDIF
```

例 7.15　下面通过 if…then…else 结构首先判断传入参数的值是否为 1，如果是则输出 1，如果不是则再判断该传入参数的值是否为 2，如果是则输出 2，当以上条件都不满足时输出 3。其代码如下。



（实例位置：光盘\TM\sl\7\7.15）

```
delimiter //
create procedure example_if(in x int)
begin
if x=1 then
select 1;
elseif x=2 then
select 2;
else
select 3;
end if;
end
//
```

以上代码的运行结果如图 7.15 所示。

```
mysql> delimiter //
mysql> create procedure example_if(in x int)
    -> begin
    -> if x=1 then
    -> select 1;
    -> elseif x=2 then
    -> select 2;
    -> else
    -> select 3;
    -> end if;
    -> end
    -> //
Query OK, 0 rows affected (0.00 sec)
```

图 7.15　应用 IF 语句的存储过程

通过 MySQL 调用该存储过程。其运行结果如图 7.16 所示。

```
mysql> call example_if(2)//
+---+
| 2 |
+---+
| 2 |
+---+
1 row in set (0.00 sec)

Query OK, 0 rows affected (0.00 sec)
```

图 7.16　调用 example_if()存储过程

7.2.2　CASE 语句

CASE 语句为多分支语句结构，该语句首先从 WHEN 后的 VALUE 中查找与 CASE 后的 VALUE

相等的值，如果查找到则执行该分支的内容，否则执行 ELSE 后的内容。CASE 语句表示形式如下。

```
CASE value
        WHEN value THEN …
        [WHEN valueTHEN…]
        [ELSE…]
END CASE
```

其中，value 参数表示条件判断的变量；WHEN…THEN 中的 value 参数表示变量的取值。

CASE 语句另一种语法表示形式如下。

```
CASE
        WHEN value THEN…
        [WHEN valueTHEN…]
        [ELSE…]
END CASE
```

例 7.16 下面通过 CASE 语句首先判断传入参数的值是否为 1，如果条件成立则输出 1，如果条件不成立则再判断该传入参数的值是否为 2，如果成立则输出 2，当以上条件都不满足时输出 3。代码如下。**（实例位置：光盘\TM\sl\7\7.16）**

```
delimiter //
create procedure example_case(in x int)
begin
case x
when 1 then select 1;
when 2 then select 2;
else select 3;
end case;
end
//
```

运行该示例的结果如图 7.17 所示。

```
mysql> delimiter //
mysql> create procedure example_case(in x int)
    -> begin
    -> case x
    -> when 1 then select 1;
    -> when 2 then select 2;
    -> else select 3;
    -> end case;
    -> end
    -> //
Query OK, 0 rows affected (0.00 sec)
```

图 7.17　应用 CASE 语句的存储过程

调用该存储过程，其运行结果如图 7.18 所示。



```
mysql> call example_case(3)//
+---+
| 3 |
+---+
| 3 |
+---+
1 row in set (0.01 sec)

Query OK, 0 rows affected (0.01 sec)
```

图 7.18　调用 example_case()存储过程

7.2.3　WHILE 循环语句

WHILE 循环语句执行时首先判断 condition 条件是否为真，如果是则执行循环体，否则退出循环。该语句表示形式如下。

```
WHILE condition DO
…
END WHILE;
```

例 7.17　下面应用 WHILE 语句求前 100 项的和。首先定义变量 i 和 s，分别用来控制循环的次数和保存前 100 项和，当变量 i 的值小于或等于 100 时，使 s 的值加 i，并同时使 i 的值增 1。直到 i 大于 100 时退出循环并输出结果。其代码如下所示。**（实例位置：光盘\TM\sl\7\7.17）**

```
delimiter //
create procedure example_while (out sum int)
begin
declare i int default 1;
declare s int default 0;
while i<=100 do
set s=s+i;
set i=i+1;
end while;
set sum=s;
end
//
```

运行以上代码的结果如图 7.19 所示。

```
mysql> delimiter //
mysql> create procedure example_while (out sum int)
    -> begin
    -> declare i int default 1;
    -> declare s int default 0;
    -> while i<=100 do
    -> set s=s+i;
    -> set i=i+1;
    -> end while;
    -> set sum=s;
    -> end
    -> //
Query OK, 0 rows affected (0.00 sec)
```

图 7.19　应用 WHILE 语句的存储过程

调用该存储过程，调用语句如下所示。

```
call example_while(@s)
mysql>select @s
```

调用该存储过程的结果如图 7.20 所示。

```
mysql> select @s//
+------+
| @s   |
+------+
| 5050 |
+------+
1 row in set (0.00 sec)

mysql> _
```

图 7.20　调用 example_while()存储过程

7.2.4　LOOP 循环语句

该循环没有内置的循环条件，但可以通过 LEAVE 语句退出循环。LOOP 语句表示形式如下。

```
LOOP
...
END LOOP
```

LOOP 允许某特定语句或语句群的重复执行，实现一个简单的循环构造，中间省略的部分是需要重复执行的语句。在循环内的语句一直重复直至循环被退出，退出循环应用 LEAVE 语句。

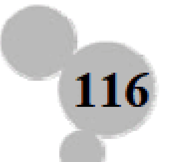
LEAVE 语句经常和 BEGIN…END 或循环一起使用，其表示形式如下。

```
LEAVE label
```

label 是语句中标注的名字，这个名字是自定义的。加上 LEAVE 关键字就可以用来退出被标注的循环语句。

例 7.18　下面应用 LOOP 语句求前 100 项的和。首先定义变量 i 和 s，分别用来控制循环的次数和保存前 100 项的和，进入该循环体后首先使 s 的值加 i，之后使 i 加 1 并进入下次循环，直到 i 大于 100，通过 LEAVE 语句退出循环并输出结果。其代码如下。**（实例位置：光盘\TM\sl\7\7.18）**

```
delimiter //
create procedure example_loop (out sum int)
begin
declare i int default 1;
declare s int default 0;
loop_label:loop
set s=s+i;
set i=i+1;
if i>100 then
```



```
leave loop_label;
end if;
end loop;
set sum=s;
end
//
```

上述代码的运行结果如图 7.21 所示。

```
mysql> delimiter //
mysql> create procedure example_loop (out sum int)
    -> begin
    -> declare i int default 1;
    -> declare s int default 0;
    -> loop_label:loop
    -> set s=s+i;
    -> set i=i+1;
    -> if i>100 then
    -> leave loop_label;
    -> end if;
    -> end loop;
    -> set sum=s;
    -> end
    -> //
Query OK, 0 rows affected (0.00 sec)
```

图 7.21　应用 LOOP 语句创建存储过程

调用名称为 example_loop 的存储过程，其代码如下。

```
call example_loop(@s)
select @s
```

运行结果如图 7.22 所示。

```
mysql> call example_loop(@s)//
Query OK, 0 rows affected (0.00 sec)

mysql> select @s//
+------+
| @s   |
+------+
| 5050 |
+------+
1 row in set (0.00 sec)
```

图 7.22　调用 example_loop()存储过程

7.2.5　REPEAT 循环语句

该语句先执行一次循环体，之后判断 condition 条件是否为真，为真则退出循环，否则继续执行循环。REPEAT 语句表示形式如下。

```
REPEAT
  ...
UNTIL condition
END REPEAT
```

例 7.19 下面应用 REPEAT 语句求前 100 项和。首先定义变量 i 和 s，分别用来控制循环的次数和保存前 100 项和，进入循环体后首先使 s 的值加 i，之后使 i 的值加 1，直到 i 大于 100 时退出循环并输出结果。**（实例位置：光盘\TM\sl\7\7.19）**

```
delimiter //
create procedure example_repeat (out sum int)
begin
declare i int default 1;
declare s int default 0;
repeat
set s=s+i;
set i=i+1;
until i>100
end repeat;
set sum=s;
end
//
```

以上代码的运行结果如图 7.23 所示。

```
mysql> delimiter //
mysql> create procedure example_repeat (out sum int)
    -> begin
    -> declare i int default 1;
    -> declare s int default 0;
    -> repeat
    -> set s=s+i;
    -> set i=i+1;
    -> until i>100
    -> end repeat;
    -> set sum=s;
    -> end
    -> //
Query OK, 0 rows affected (0.00 sec)
```

图 7.23　应用 REPEAT 语句创建存储过程

调用该存储过程，相关代码如下所示。

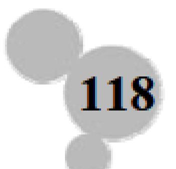
```
call example_repeat(@s)
select @s
```

调用该存储过程的运行结果如图 7.24 所示。

```
mysql> call example_repeat(@s)//
Query OK, 0 rows affected (0.02 sec)

mysql> select @s//
+------+
| @s   |
+------+
| 5050 |
+------+
1 row in set (0.00 sec)
```

图 7.24　调用 example_repeat()存储过程



循环语句中还有一个 ITERATE 语句，它可以出现在 LOOP、REPEAT 和 WHILE 语句内，其意为“再次循环”。该语句格式如下。

```
ITERATE label
```

该语句的格式与 LEAVE 大同小异，区别在于：LEAVE 语句是离开一个循环，而 ITERATE 语句是重新开始一个循环。

注意

与一般程序设计流程控制不同的是：存储过程并不支持 FOR 循环。

7.3 小 结

本章对 MySQL 的运算符和流程控制语句进行了详细讲解，并通过举例说明，帮助读者更好地理解所学知识的用法。在阅读本章时，读者应该重点掌握各种运算符和流程控制语句的使用，其中，位运算符是本章的难点。因为，位运算符需要将操作数转换为二进制数，然后进行位运算。这要求读者能够掌握二进制运算的相关知识。

7.4 实践与练习

1. 编写 SQL 语句，将数字 2、0 和 null 之间的任意两个进行逻辑运算。**（答案位置：光盘\TM\sl\7\7.20）**
2. 应用 WHILE 语句求前 10 项的和。**（答案位置：光盘\TM\sl\7\7.21）**

第 8 章

表数据的增、删、改操作

（视频讲解：20 分钟）

成功创建数据库和数据表以后，就可以针对表中的数据进行各种交互操作了。这些操作可以有效地使用、维护和管理数据库中的表数据，其中最常用的就是添加、修改和删除操作了。本章将详细介绍如何通过 SQL 语句来实现表数据的增、删和改操作。

通过阅读本章，读者可以：

- 掌握插入表数据的方法
- 掌握修改表中数据的方法
- 掌握删除表数据的两种方法

8.1　插 入 数 据

在建立一个空的数据库和数据表时，首先需要考虑的是如何向数据表中添加数据，该操作可以使用 INSERT 语句来完成。使用 INSERT 语句可以向一个已有数据表中插一个新行，也就是插入一行新记录。在 MySQL 中，INSERT 语句有 3 种语法格式，分别是 INSERT…VALUES 语句、INSERT…SET 语句和 INSERT…SELECT 语句。下面将分别进行介绍。

8.1.1　使用 INSERT…VALUES 语句插入数据

使用 INSERT…VALUES 语句插入数据，是 INSERT 语句的最常用的语法格式。它的语法格式如下。

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
    [INTO] 数据表名 [(字段名,…)]
    VALUES ({值 | DEFAULT},…),(…),…
    [ ON DUPLICATE KEY UPDATE 字段名=表达式,…]
```

参数说明如下。

（1）[LOW_PRIORITY|DELAYED|HIGH_PRIORITY]：可选项，其中，LOW_PRIORITY 是 INSERT、UPDATE 和 DELETE 语句都支持的一种可选修饰符，通常应用在多用户访问数据库的情况下，用于指示 MySQL 降低 INSERT、DELETE 或 UPDATE 操作执行的优先级；DELAYED 是 INSERT 语句支持的一种可选修饰符，用于指定 MySQL 服务器把待插入的行数据放到一个缓冲器中，直到待插数据的表空闲时，才真正在表中插入数据行；HIGH_PRIORITY 是 INSERT 和 SELECT 语句支持的一种可选修饰符，用于指定 INSERT 和 SELECT 操作优先执行的。

（2）[IGNORE]：可选项，表示在执行 INSERT 语句时，所出现的错误都会被当作警告处理。

（3）[INTO] 数据表名：可选项，用于指定被操作的数据表。

（4）[(字段名,…)]：可选项，当不指定该选项时，表示要向表中所有列插入数据，否则表示向数据表的指定列插入数据。

（5）VALUES ({值 | DEFAULT},…),(…),…：必选项，用于指定需要插入的数据清单，其顺序必须与字段的顺序相对应。其中的每一列的数据可以是一个常量、变量、表达式或者 NULL，但是其数据类型要与对应的字段类型相匹配；也可以直接使用 DEFAULT 关键字，表示为该列插入默认值，但是使用的前提是已经明确指定了默认值，否则会出错。

（6）ON DUPLICATE KEY UPDATE 子句：可选项，用于指定向表中插入行时，如果导致 UNIQUE KEY 或 PRIMARY KEY 出现重复值，系统会根据 UPDATE 后的语句修改表中原有行数据。

INSERT…VALUES 语句在使用时，通常有以下 3 种方式。

1．插入完整数据

例 8.1 通过 INSERT…VALUES 语句向数据表 tb_admin 中插入一条完整的数据。**（实例位置：光盘\TM\sl\8\8.1）**

（1）在编写 SQL 语句之前，先查看一下数据表 tb_admin 的表结构，具体代码如下。

```
DESC db_database08.tb_admin;
```

运行效果如图 8.1 所示。

```
mysql> DESC db_database08.tb_admin;
+------------+-------------+------+-----+---------+----------------+
| Field      | Type        | Null | Key | Default | Extra          |
+------------+-------------+------+-----+---------+----------------+
| id         | int(11)     | NO   | PRI | NULL    | auto_increment |
| user       | varchar(30) | NO   |     | NULL    |                |
| password   | varchar(30) | NO   |     | NULL    |                |
| createtime | datetime    | YES  |     | NULL    |                |
+------------+-------------+------+-----+---------+----------------+
4 rows in set (0.02 sec)

mysql> _
```

图 8.1 查看数据表 tb_admin 的表结构

（2）编写 SQL 语句，先选择数据表所在的数据库，然后再应用 INSERT…VALUES 语句实现向数据表 tb_admin 中插入一条完整的数据，具体代码如下。

```
USE db_database08;
INSERT INTO tb_admin VALUES(1,'mr','mrsoft','2014-09-05 10:25:20');
```

运行效果如图 8.2 所示。

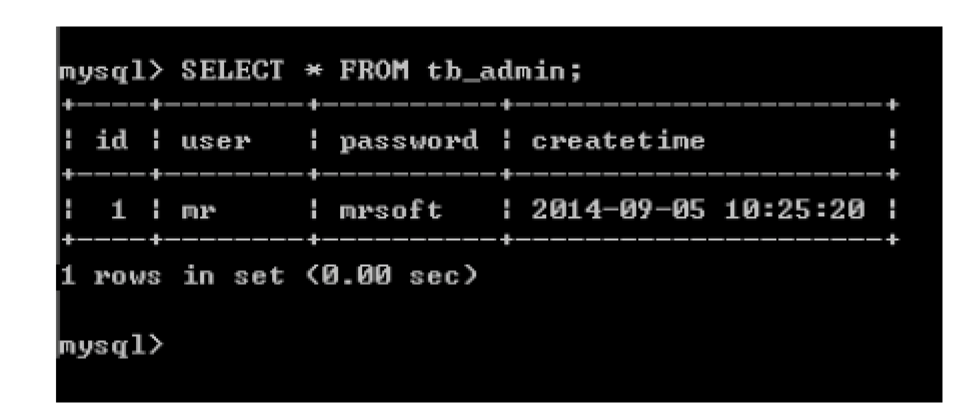


图 8.2 向数据表 tb_admin 中插入一条完整的数据

（3）通过 SELECT * FROM tb_admin 语句来查看数据表 tb_admin 中的数据，具体代码如下。

```
SELECT * FROM tb_admin;
```

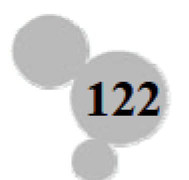
执行效果如图 8.3 所示。



```
mysql> SELECT * FROM tb_admin;
+----+------+----------+---------------------+
| id | user | password | createtime          |
+----+------+----------+---------------------+
|  1 | mr   | mrsoft   | 2014-09-05 10:25:20 |
+----+------+----------+---------------------+
1 rows in set (0.00 sec)

mysql>
```

图 8.3 查看新插入的数据



2. 插入数据记录的一部分

通过 INSERT…VALUES 语句还可以实现向数据表中插入数据记录的一部分，也就是只插入表的一行中的某几个字段的值，下面通过一个具体的实例来演示如何向数据表中插入数据记录的一部分。还是以例 8.1 中使用的数据表 tb_admin 为例进行插入。

例 8.2　通过 INSERT…VALUES 语句向数据表 tb_admin 中插入数据记录的一部分。**（实例位置：光盘\TM\sl\8\8.2）**

（1）编写 SQL 语句，先选择数据表所在的数据库，然后再应用 INSERT…VALUES 语句实现向数据表 tb_admin 中插入一条记录，只包括 user 和 password 字段的值，具体代码如下。

```
USE db_database08;
INSERT INTO tb_admin (user,password)
     VALUES('rjkflm','111');
```

运行效果如图 8.4 所示。

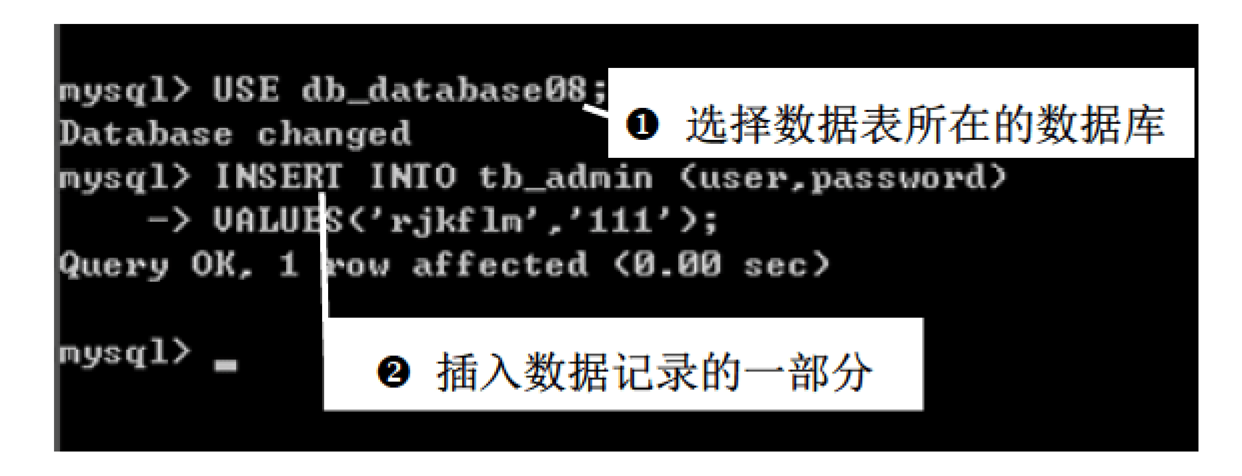


图 8.4　向数据表 tb_admin 中插入数据记录的一部分

（2）通过 SELECT * FROM tb_admin 语句来查看数据表 tb_admin 中的数据，具体代码如下。

```
SELECT * FROM tb_admin;
```

执行效果如图 8.5 所示。

```
mysql> SELECT * FROM tb_admin;
+----+--------+----------+---------------------+
| id | user   | password | createtime          |
+----+--------+----------+---------------------+
|  1 | mr     | mrsoft   | 2014-09-05 10:25:20 |
|  2 | rjkflm | 111      | NULL                |
+----+--------+----------+---------------------+
2 rows in set (0.00 sec)

mysql>
```

id 为 2 的行是新插入的数据

图 8.5　查看新插入的数据



由于在设计数据表时，将 id 字段设置为自动编号，所以即使没有指定 id 的值，MySQL 也会自动为它填上相应的编号。

3．插入多条记录

通过 INSERT…VALUES 语句还可以实现一次性插入多条数据记录。使用该方法批量插入数据，比使用多条单行的 INSERT 语句的效率要高。下面将通过一个具体的实例演示如何一次插入多条记录。

例 8.3 通过 INSERT…VALUES 语句向数据表 tb_admin 中一次插入多条记录。（**实例位置：光盘\TM\sl\8\8.3**）

（1）编写 SQL 语句，先选择数据表所在的数据库，然后再应用 INSERT…VALUES 语句实现向数据表 tb_admin 中插入 3 条记录，都只包括 user、password 和 createtime 字段的值，具体代码如下。

```
USE db_database08;
INSERT INTO tb_admin (user,password,createtime)
      VALUES('mrbccd','111', '2014-09-05 10:35:26')
      ,( 'mingri','111', '2014-09-05 10:45:27')
      ,( 'mingrisoft','111', '2014-09-05 10:55:28');
```

运行效果如图 8.6 所示。

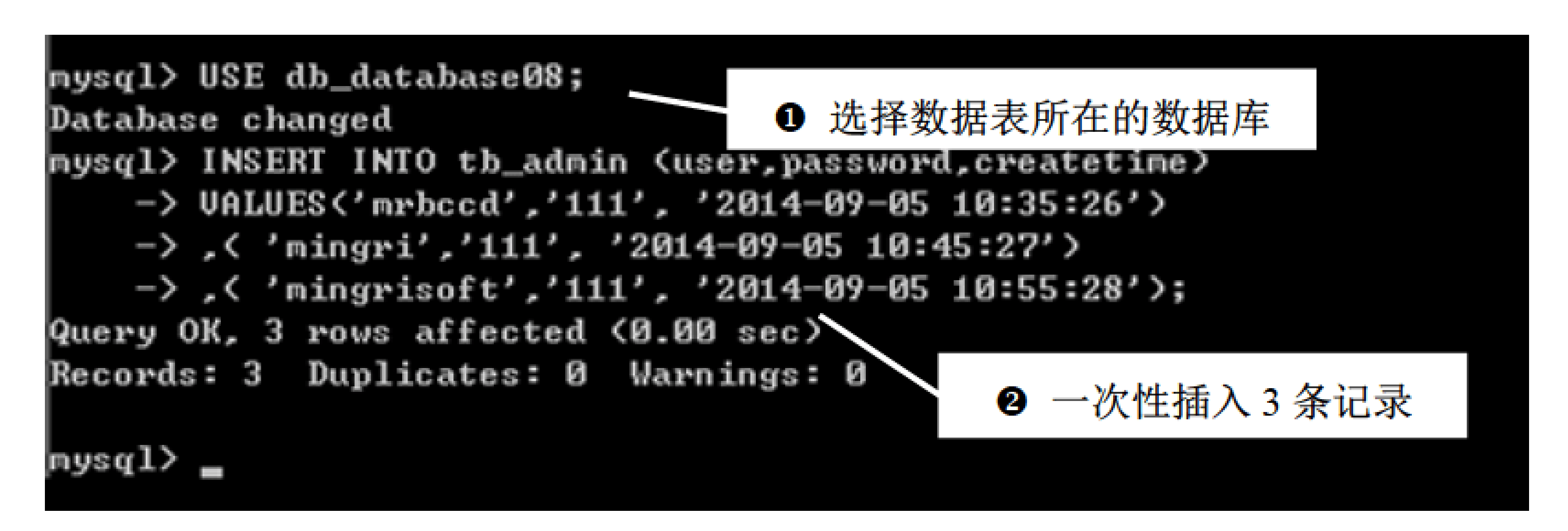


图 8.6　向数据表 tb_admin 中插入 3 条记录

（2）通过 SELECT * FROM tb_admin 语句来查看数据表 tb_admin 中的数据，具体代码如下。

```
SELECT * FROM tb_admin;
```

执行效果如图 8.7 所示。

```
mysql> SELECT * FROM tb_admin;
+----+------------+----------+---------------------+
| id | user       | password | createtime          |
+----+------------+----------+---------------------+
|  1 | mr         | mrsoft   | 2014-09-05 10:25:20 |
|  2 | rjkflm     | 111      | NULL                |
|  3 | mrbccd     | 111      | 2014-09-05 10:35:26 |
|  4 | mingri     | 111      | 2014-09-05 10:45:27 |
|  5 | mingrisoft | 111      | 2014-09-05 10:55:28 |
+----+------------+----------+---------------------+
5 rows in set (0.00 sec)

mysql>
```

新插入的 3 行数据

图 8.7　查看新插入的 3 行数据



8.1.2　使用 INSERT…SET 语句插入数据

在 MySQL 中，除了可以使用 INSERT…VALUES 语句插入数据外，还可以使用 INSERT…SET 语句插入数据。这种语法格式用于通过直接给表中的某些字段指定对应的值来实现插入指定数据，对于未指定值的字段将采用默认值进行添加。INSERT…SET 语句的语法格式如下。

```
INSERT [LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]
    [INTO] 数据表名
    SET 字段名={值 | DEFAULT}, …
    [ ON DUPLICATE KEY UPDATE 字段名=表达式,…]
```

参数说明如下。

（1）[LOW_PRIORITY | DELAYED | HIGH_PRIORITY] [IGNORE]：可选项，其作用与 INSERT…VALUES 语句相同，这里将不再赘述。

（2）[INTO] 数据表名：用于指定被操作的数据表，其中，[INTO]为可选项，可以省略。

（3）SET 字段名={值 | DEFAULT}：用于给数据表中的某些字段设置要插入的值。

（4）ON DUPLICATE KEY UPDATE 子句：可选项，其作用与 INSERT…VALUES 语句相同，这里将不再赘述。

例 8.4　通过 INSERT…SET 语句向数据表 tb_admin 中插入一条记录。**（实例位置：光盘\TM\sl\8\8.4）**

（1）编写 SQL 语句，先选择数据表所在的数据库，然后再应用 INSERT…SET 语句实现向数据表 tb_admin 中插入一条记录，包括 user、password 和 createtime 字段的值，具体代码如下。

```
USE db_database08;
INSERT INTO tb_admin
    SET user='mrbccd',password='111',createtime= '2014-09-06 10:35:26';
```

运行效果如图 8.8 所示。

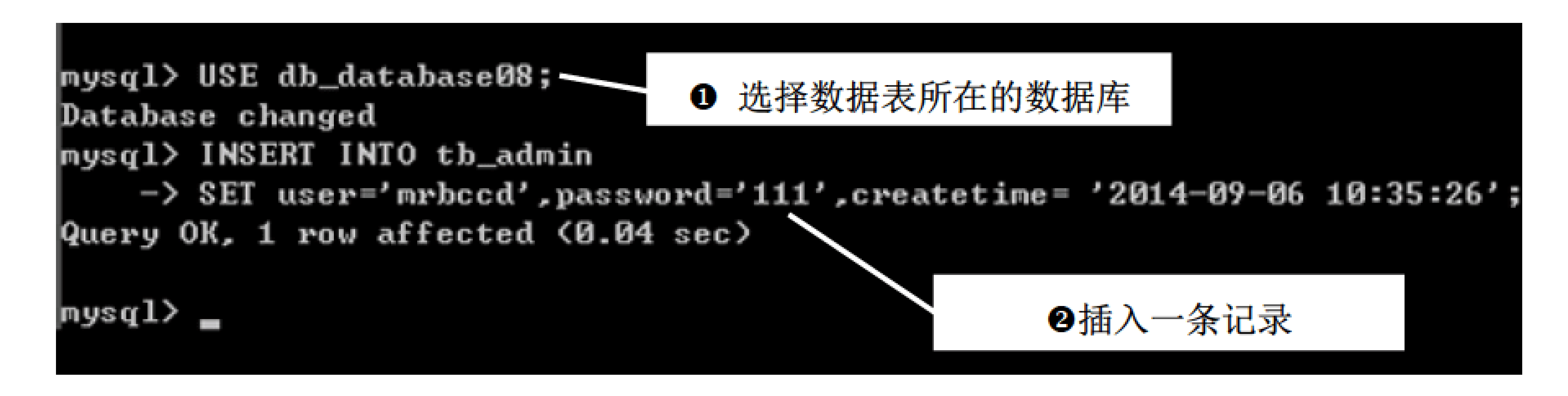


图 8.8　向数据表 tb_admin 中插入一条记录

（2）通过 SELECT * FROM tb_admin 语句来查看数据表 tb_admin 中的数据，具体代码如下。

```
SELECT * FROM tb_admin;
```

执行效果如图 8.9 所示。

```
mysql> SELECT * FROM tb_admin;
+----+-------------+----------+---------------------+
| id | user        | password | createtime          |
+----+-------------+----------+---------------------+
|  1 | mr          | mrsoft   | 2014-09-05 10:25:20 |
|  2 | rjkflm      | 111      | NULL                |
|  3 | mrbccd      | 111      | 2014-09-05 10:35:26 |
|  4 | mingri      | 111      | 2014-09-05 10:45:27 |
|  5 | mingrisoft  | 111      | 2014-09-05 10:55:28 |
|  6 | mrbccd      | 111      | 2014-09-06 10:35:26 |
+----+-------------+----------+---------------------+
6 rows in set (0.10 sec)

mysql> _
```

新插入的一行数据

图 8.9　查看新插入的一行数据

8.1.3　插入查询结果

在 MySQL 中，支持将查询结果插入到指定的数据表中，这可以通过 INSERT…SELECT 语句来实现。

```
INSERT [LOW_PRIORITY | HIGH_PRIORITY] [IGNORE]
    [INTO] 数据表名 [(字段名,…)]
    SELECT …
    [ ON DUPLICATE KEY UPDATE 字段名=表达式, … ]
```

参数说明如下。

（1）[LOW_PRIORITY|DELAYED|HIGH_PRIORITY] [IGNORE]：可选项，其作用与 INSERT…VALUES 语句相同，这里不再赘述。

（2）[INTO] 数据表名：用于指定被操作的数据表，其中，[INTO]为可选项，可以省略。

（3）[(字段名,…)]：可选项，当不指定该选项时，表示要向表中所有列插入数据，否则表示向数据表的指定列插入数据。

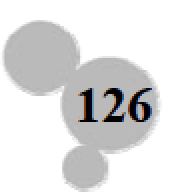
（4）SELECT 子句：用于快速地从一个或者多个表中取出数据，并将这些数据作为行数据插入到目标数据表中。需要注意的是，SELECT 子句返回的结果集中的字段数、字段类型必须与目标数据表完全一致。

（5）ON DUPLICATE KEY UPDATE 子句：可选项，其作用与 INSERT…VALUES 语句相同，这里不再赘述。

例 8.5　从数据表 tb_mrbook 中查询出 user 和 pass 字段的值，插入到数据表 tb_admin 中。**（实例位置：光盘\TM\sl\8\8.5）**

（1）查看数据表 tb_mrbook 的表结构，具体代码如下。

```
DESC db_database08.tb_mrbook;
```



执行效果如图 8.10 所示。

```
mysql> DESC db_database08.tb_mrbook;
+----------+-------------+------+-----+---------+----------------+
| Field    | Type        | Null | Key | Default | Extra          |
+----------+-------------+------+-----+---------+----------------+
| id       | int(11)     | NO   | PRI | NULL    | auto_increment |
| user     | varchar(40) | YES  |     | NULL    |                |
| pass     | varchar(20) | YES  |     | NULL    |                |
| bookname | varchar(50) | YES  |     | NULL    |                |
| type     | varchar(20) | YES  |     | NULL    |                |
| price    | varchar(20) | YES  |     | NULL    |                |
+----------+-------------+------+-----+---------+----------------+
6 rows in set (0.02 sec)

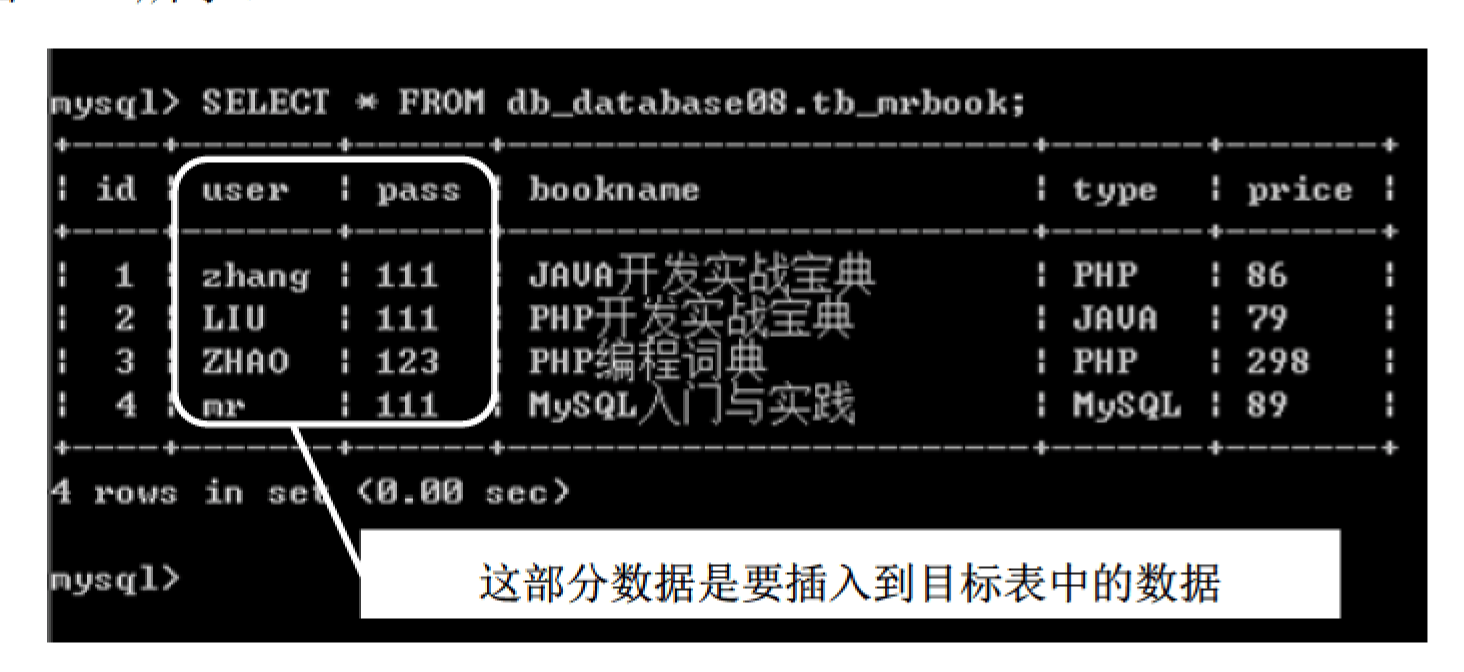
mysql>
```

图 8.10　数据表 tb_mrbook 的表结构

（2）查询数据表 tb_mrbook 中的数据，具体代码如下。

```
SELECT * FROM tb_mrbook;
```

执行效果如图 8.11 所示。



```
mysql> SELECT * FROM db_database08.tb_mrbook;
+----+-------+------+-------------------+-------+-------+
| id | user  | pass | bookname          | type  | price |
+----+-------+------+-------------------+-------+-------+
|  1 | zhang | 111  | JAVA开发实战宝典  | PHP   | 86    |
|  2 | LIU   | 111  | PHP开发实战宝典   | JAVA  | 79    |
|  3 | ZHAO  | 123  | PHP编程词典       | PHP   | 298   |
|  4 | mr    | 111  | MySQL入门与实践   | MySQL | 89    |
+----+-------+------+-------------------+-------+-------+
4 rows in set (0.00 sec)

mysql>
```


图 8.11　数据表 tb_mrbook 的数据

（3）编写 SQL 语句，实现从数据表 tb_mrbook 中查询 user 和 pass 字段的值，插入到数据表 tb_admin 中，具体代码如下。

```
INSERT INTO db_database08.tb_admin
      (user,password)
      SELECT user,pass FROM tb_mrbook;
```

执行效果如图 8.12 所示。

```
mysql> INSERT INTO db_database08.tb_admin
    -> (user,password)
    -> SELECT user,pass FROM tb_mrbook;
Query OK, 4 rows affected (0.04 sec)
Records: 4  Duplicates: 0  Warnings: 0

mysql> _
```

图 8.12　将查询结果插入到数据表

（4）通过 SELECT 语句来查看数据表 tb_admin 中的数据，具体代码如下。

```
SELECT * FROM db_database08.tb_admin;
```

执行效果如图 8.13 所示。

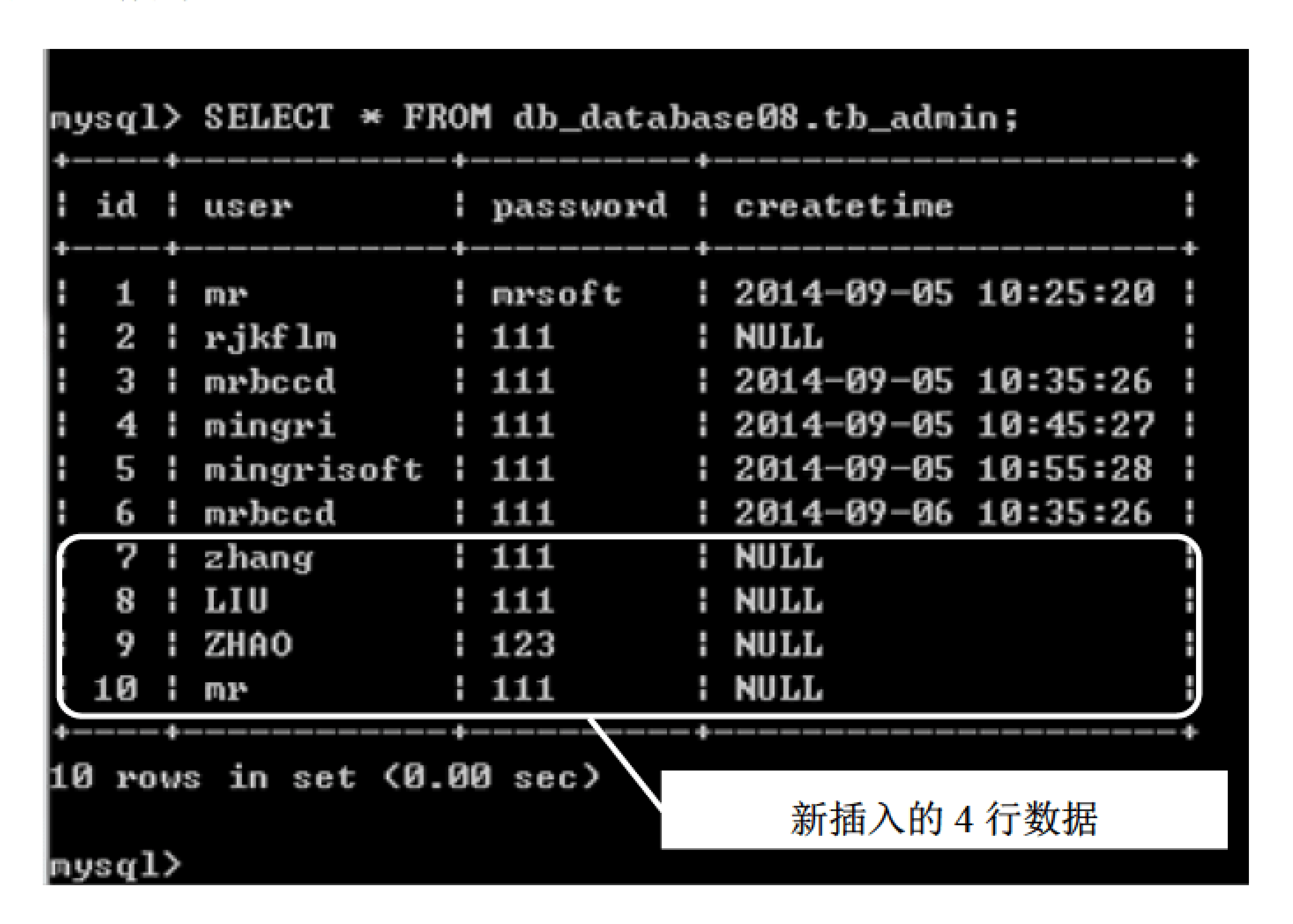


图 8.13　查看新插入的数据

8.2　修改数据

要执行修改的操作可以使用 UPDATE 语句，语法如下。

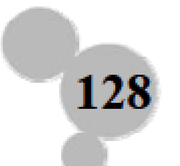
```
UPDATE [LOW_PRIORITY] [IGNORE] 数据表名
    SET 字段 1=值 1 [, 字段 2=值 2…]
    [WHERE 条件表达式]
    [ORDER BY…]
    [LIMIT 行数]
```

参数说明如下。

（1）[LOW_PRIORITY]：可选项，表示在多用户访问数据库的情况下可用于延迟 UPDATE 操作，直到没有别的用户再从表中读取数据为止。这个过程仅适用于表级锁的存储引擎（如 IyISAM、MEMORY 和 MERGE）。

（2）[IGNORE]：在 MySQL 中，通过 UPDATE 语句更新表中多行数据时，如果出现错误，那么整个 UPDATE 语句操作都会被取消，错误发生前更新的所有行将被恢复到它们原来的值。因此，为了在发生错误时也要继续进行更新，则可以在 UPDATE 语句中使用 IGNORE 关键字。

（3）SET 子句：必选项，用于指定表中要修改的字段名及其字段值。其中的值可以是表达式，也可以是该字段所对应的默认值。如果指定默认值，那么使用关键字 DEFAULT 指定。



（4）WHERE 子句：可选项，用于限定表中要修改的行，如果不指定该子句，那么 UPDATE 语句会更新表中的所有行。

（5）ORDER BY 子句：可选项，用于限定表中的行被修改的次序。

（6）LIMIT 子句：可选项，用于限定被修改的行数。

例 8.6　将管理员信息表 tb_admin 中用户名为 mrbccd 的管理员密码 111 修改为 123。**（实例位置：光盘\TM\sl\8\8.6）**

编写 SQL 语句修改用户名为 mrbccd 的管理员密码为 123，具体代码如下。

```
UPDATE db_database08.tb_admin SET password='123' WHERE user='mrbccd';
```

运行效果如图 8.14 所示。

```
mysql> UPDATE db_database08.tb_admin SET password='123' WHERE user='mrbccd';
Query OK, 2 rows affected (0.02 sec)
Rows matched: 2  Changed: 2  Warnings: 0

mysql> _
```

将 mrbccd 用户的密码更改为 123

图 8.14　将 mrbccd 用户的密码更改为 123

注意

更新时一定要保证 WHERE 子句的正确性，一旦 WHERE 子句出错，将会破坏所有改变的数据。

查询修改后的数据库内容，代码如下。

```
SELECT * FROM db_database08.tb_admin WHERE user='mrbccd';
```

执行结果如图 8.15 所示。

```
mysql> SELECT * FROM db_database08.tb_admin WHERE user='mrbccd';
+----+--------+----------+---------------------+
| id | user   | password | createtime          |
+----+--------+----------+---------------------+
|  3 | mrbccd | 123      | 2014-09-05 10:35:26 |
|  6 | mrbccd | 123      | 2014-09-06 10:35:26 |
+----+--------+----------+---------------------+
2 rows in set (0.00 sec)

mysql>
```

图 8.15　查看修改后的结果

8.3　删除数据

在数据库中，有些数据已经失去意义或者错误时就需要将它们删除，在 MySQL 中，可以使用 DELETE 语句或者 TRUNCATE TABLE 语句删除表中的一行或多行数据，下面分别进行介绍。

8.3.1　通过 DELETE 语句删除数据

通过 DELETE 语句删除数据的基本语法格式如下。

```
DELETE [LOW_PRIORITY] [QUICK] [IGNORE] FROM 数据表名
    [WHERE 条件表达式]
    [ORDER BY…]
    [LIMIT 行数]
```

参数说明如下。

（1）[LOW_PRIORITY]：可选项，表示在多用户访问数据库的情况下可用于延迟 DELETE 操作，直到没有别的用户再从表中读取数据为止。这个过程仅适用于表级锁的存储引擎（如 IyISAM、MEMORY 和 MERGE）。

（2）[QUICK]：可选项，用于加快部分种类的删除操作的速度。

（3）[IGNORE]：在 MySQL 中，通过 DELETE 语句删除表中多行数据时，如果出现错误，那么整个 DELETE 语句操作都会被取消，错误发生前更新的所有行将被恢复到它们原来的值。因此，为了在发生错误时也要继续进行删除，则可以在 DELETE 语句中使用 IGNORE 关键字。

（4）数据表名：用于指定要删除的数据表的表名。

（5）WHERE 子句：可选项，用于限定表中要删除的行，如果不指定该子句，那么 DELETE 语句会删除表中的所有行。

（6）ORDER BY 子句：可选项，用于限定表中的行被删除的次序。

（7）LIMIT 子句：可选项，用于限定被删除的行数。

注意

该语句在执行过程中，如果没有指定 WHERE 条件，将删除所有的记录；如果指定了 WHERE 条件，将按照指定的条件进行删除。

例 8.7　删除管理员数据表 tb_admin 中用户名为 mr 的记录信息。（**实例位置：光盘\TM\sl\8\8.7**）

（1）编写 SQL 语句除管理员数据表 tb_admin 中用户名为 mr 的记录信息，具体代码如下。

```
USE db_database08;
DELETE FROM tb_admin WHERE user='mr';
```

运行效果如图 8.16 所示。

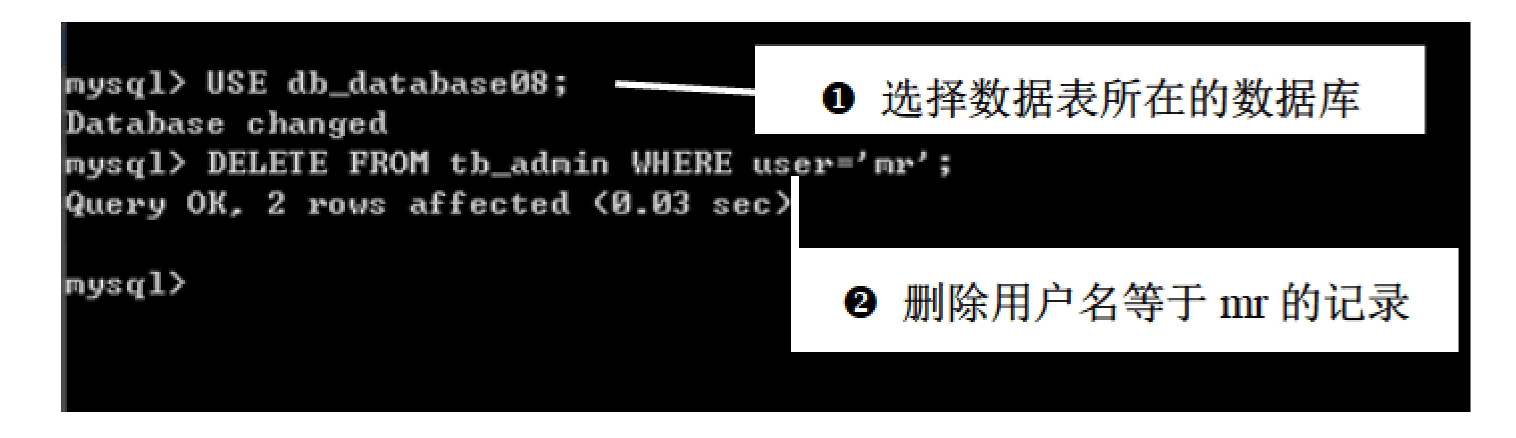
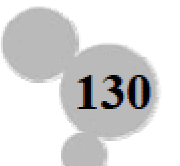


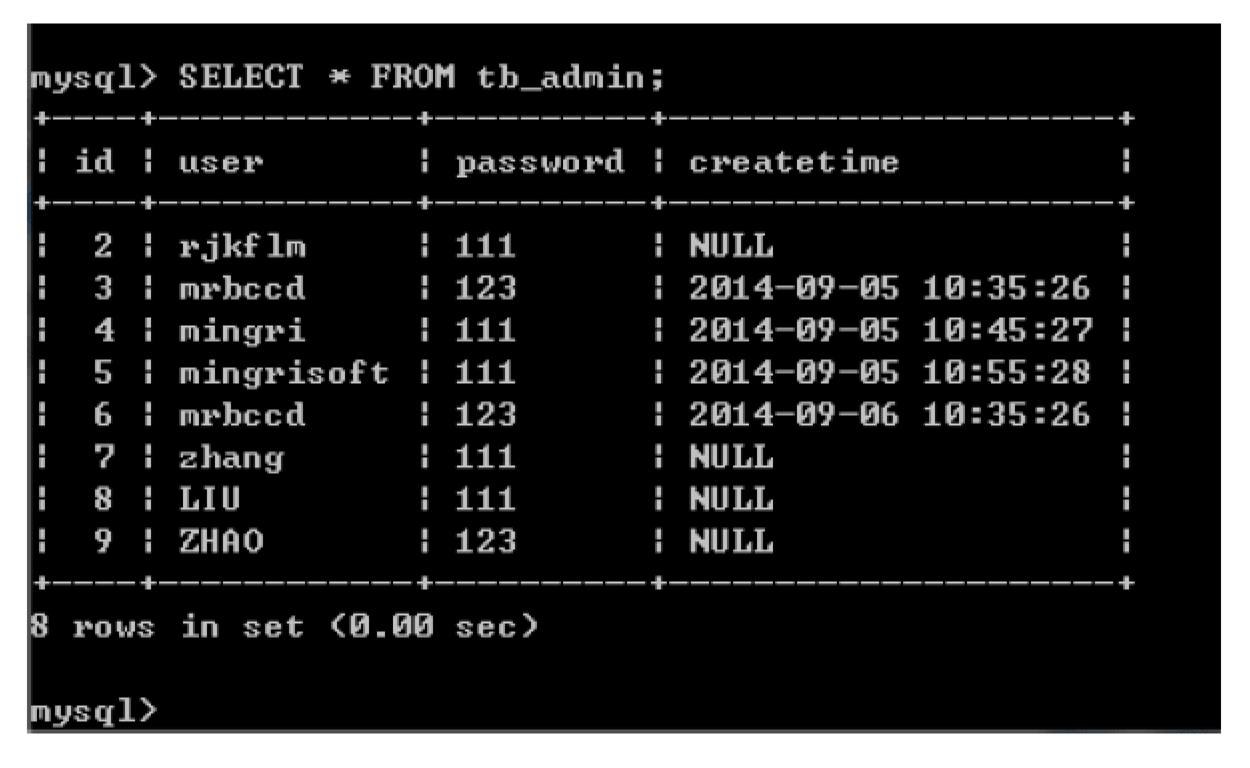
图 8.16　删除数据表中用户名为 mr 的记录



（2）通过 SELECT 语句来查看删除记录后数据表 tb_admin 中的数据，具体代码如下。

```
SELECT * FROM tb_admin;
```

执行结果如图 8.17 所示。



```
mysql> SELECT * FROM tb_admin;
+----+------------+----------+---------------------+
| id | user       | password | createtime          |
+----+------------+----------+---------------------+
|  2 | rjkflm     | 111      | NULL                |
|  3 | mrbccd     | 123      | 2014-09-05 10:35:26 |
|  4 | mingri     | 111      | 2014-09-05 10:45:27 |
|  5 | mingrisoft | 111      | 2014-09-05 10:55:28 |
|  6 | mrbccd     | 123      | 2014-09-06 10:35:26 |
|  7 | zhang      | 111      | NULL                |
|  8 | LIU        | 111      | NULL                |
|  9 | ZHAO       | 123      | NULL                |
+----+------------+----------+---------------------+
8 rows in set (0.00 sec)

mysql>
```

图 8.17　查看删除后的结果



删除数据前，数据表 tb_admin 中的数据如图 8.13 所示。

注意

在实际的应用中，执行删除操作时，执行删除的条件一般应该为数据的 id，而不是具体某个字段值，这样可以避免一些错误发生。

8.3.2　通过 TRUNCATE TABLE 语句删除数据

在删除数据时，如果要从表中删除所有的行，通过 TRUNCATE TABLE 语句删除数据的基本语法格式如下。

```
TRUNCATE [TABLE] 数据表名
```

在上面的语法中，数据表名表示的就是删除的数据表的表名，也可以使用“数据库名.数据表名”来指定该数据表隶属于哪个数据库。

注意

由于 TRUNCATE TABLE 语句会删除数据表中的所有数据，并且无法恢复，因此使用 TRUNCATE TABLE 语句时一定要十分小心。

例 8.8　使用 TRUNCATE TABLE 语句清空管理员数据表 tb_admin，具体代码如下。**（实例位置：**

光盘\TM\sl\8\8.8）

```
TRUNCATE TABLE db_database08.tb_admin;
```

运行效果如图 8.18 所示。

```
mysql> TRUNCATE TABLE db_database08.tb_admin;
Query OK, 0 rows affected (0.03 sec)

mysql> _
```

图 8.18　清空管理员数据表 tb_admin

DELETE 语句和 TRUNCATE TABLE 语句的区别如下。

（1）使用 TRUNCATE TABLE 语句后，表中的 AUTO_INCREMENT 计数器将被重新设置为该列的初始值。

（2）对于参与了索引和视图的表，不能使用 TRUNCATE TABLE 语句来删除数据，而应用使用 DELETE 语句。

（3）TRUNCATE TABLE 操作与 DELETE 操作使用的系统和事务日志资源少。DELETE 语句每删除一行都会在事务日志中添加一行记录，而 TRUNCATE TABLE 语句是通过释放存储表数据所用的数据页来删除数据的，因此只在事务日志中记录页的释放。

8.4　小　　结

本章介绍了在 MySQL 中向数据表中添加数据库、修改数据和删除数据的具体方法，也就是对表数据的增、删和改操作。这 3 种操作在实际开发中经常应用。因此，对于本章的内容需要认真学习，争取做到举一反三、灵活应用。

8.5　实践与练习

1．编写 SQL 语句，先选择数据表所在的数据库，然后再应用 INSERT…VALUES 语句实现向数据表 tb_admin 中插入一条记录，只包括 user、password 和 createtime 字段的值。**（答案位置：光盘\TM\sl\8\8.9）**

2．编写 SQL 语句，使用 DELETE 语句清空管理员数据表 tb_admin。**（答案位置：光盘\TM\sl\8\8.10）**

第 9 章

数据查询

（视频讲解：52 分钟）

数据查询是指从数据库中获取所需要的数据。数据查询是数据库操作中最常用，也是最重要的操作。通过不同的查询方式可以获得不同的数据。用户可以根据自己对数据的需求使用不同的查询方式。在 MySQL 中是使用 SELECT 语句来查询数据的。本章将对查询语句的基本语法、在单表上查询数据、使用聚合函数查询数据、合并查询结果等内容进行详细的讲解，帮助读者轻松了解查询数据的语句。

通过阅读本章，读者可以：

- 了解基本查询语句
- 了解单表查询的方法
- 了解用聚合函数查询的方法
- 掌握连接查询和子查询的方法
- 掌握合并查询结果的方法
- 掌握定义表和字段别名的方法
- 掌握正则表达式查询的方法

9.1 基本查询语句

SELECT 语句是最常用的查询语句，它的使用方式有些复杂，但功能是相当强大的。SELECT 语句的基本语法如下。

```
select selection_list                    //要查询的内容，选择哪些列
from 数据表名                            //指定数据表
where primary_constraint                 //查询时需要满足的条件，行必须满足的条件
group by grouping_columns                //如何对结果进行分组
order by sorting_cloumns                 //如何对结果进行排序
having secondary_constraint              //查询时满足的第二条件
limit count                              //限定输出的查询结果
```

其中使用的子句将在后面逐个介绍。下面先介绍 SELECT 语句的简单应用。

1. 使用 SELECT 语句查询一个数据表

使用 SELECT 语句时，首先要确定所要查询的列。“*”代表所有的列。例如，查询 db_database09 数据库 user 表中的所有数据，代码如下。

```
mysql> use db_database09
Database changed
mysql> select * from user;
+----+------+----------+--------------+
| id | user | lxdh     | jtdz         |
+----+------+----------+--------------+
|  1 | mr   | 12345678 | 长春市 |
|  2 | mrsoft | 87654321 | 四平市     |
+----+------+----------+--------------+
2 rows in set (0.00 sec)
```

这是查询整个表中所有列的操作，还可以针对表中的某一列或多列进行查询。

2. 查询表中的一列或多列

针对表中的多列进行查询，只要在 select 后面指定要查询的列名即可，多列之间用“,”分隔。例如，查询 user 表中的 id 和 lxdh，代码如下。

```
mysql> select id, lxdh from user ;
+----+----------+
| id | lxdh     |
+----+----------+
|  1 | 12345678 |
|  2 | 87654321 |
+----+----------+
2 rows in set (0.00 sec)
```

3. 从一个或多个表中获取数据

使用 SELECT 语句进行查询，需要确定所要查询的数据在哪个表或哪些表中，在对多个表进行查询时，同样使用“,”对多个表进行分隔。

例 9.1 从 tb_admin 表和 tb_students 表中查询出 tb_admin.id、tb_admin.tb_user、tb_students.id 和 tb_students.name 字段的值，其代码如下。**（实例位置：光盘\TM\sl\9\9.1）**

```
mysql> select tb_admin.id,tb_admin.tb_user,tb_students.id,tb_students.name from
tb_admin,tb_students;
+----+----------+----+------+
| id | tb_user   | id | name |
+----+----------+----+------+
|   1 | mr           |   1 |  潘攀        |
|   2 |  明日科技          |   1 |  潘攀        |
+----+----------+----+------+
2 rows in set (0.03 sec)
```

说明

在查询数据库中的数据时，如果数据中涉及中文字符串，有可能在输出时会出现乱码。那么最后在执行查询操作之前，通过 set names 语句设置其编码格式，然后再输出中文字符串时就不会出现乱码了。如本例中所示，应用 set names 语句设置其编码格式为 gb2312。

还可以在 WHERE 子句中使用连接运算来确定表之间的联系，然后根据这个条件返回查询结果。例如，从家庭收入表（jtsr）中查询出指定用户的家庭收入数据，条件是用户的 ID 为 1，代码如下。

```
mysql> select jtsr from user,jtsr
    -> where   user.user=jtsr.user and user.id=1 ;
+------+
| jtsr |
+------+
| 10000 |
+------+
2 rows in set (0.00 sec)
```

其中，user.user = jtsr.user 将表 user 和 jtsr 连接起来，叫作等同连接；如果不使用 user.user= jtsr.user，那么产生的结果将是两个表的笛卡尔积，叫作全连接。

9.2　单 表 查 询

单表查询是指从一张表中查询所需要的数据，所有查询操作都比较简单。

9.2.1　查询所有字段

查询所有字段是指查询表中所有字段的数据。这种方式可以将表中所有字段的数据都查询出来。在 MySQL 中可以使用“*”代表所有的列，即可查出所有的字段，语法格式如下。

```
SELECT * FROM 表名;
```

其应用已经在 9.1 节基本查询语句中介绍过，这里不再赘述。

9.2.2　查询指定字段

查询指定字段可以使用下面的语法格式。

```
SELECT 字段名 FROM 表名;
```

如果是查询多个字段，可以使用“,”对字段进行分隔。

例 9.2　查询 db_database09 数据库 tb_login 表中 user 和 pwd 两个字段，SELECT 查询语句如下。**（实例位置：光盘\TM\sl\9\9.2）**

```
SELECT user,pwd FROM tb_login;
```

查询结果如图 9.1 所示。

```
mysql> select user,pwd from tb_login;
+------+---------+
| user | pwd     |
+------+---------+
| mrkj | mrkj    |
| lx   | lx      |
| mr   | mrsoft  |
+------+---------+
3 rows in set (0.00 sec)

mysql>
```

图 9.1　查询指定字段的数据

9.2.3　查询指定数据

如果要从很多记录中查询出指定的记录，那么就需要一个查询的条件。设定查询条件应用的是 WHERE 子句。通过它可以实现很多复杂的条件查询。在使用 WHERE 子句时，需要使用一些比较运算符来确定查询的条件。其常用的比较运算符如表 9.1 所示。

表 9.1　比较运算符

运　算　符	名　　称	示　　例	运　算　符	名　　称	示　　例
=	等于	Id=5	Is not null	n/a	Id is not null
>	大于	Id>5	Between	n/a	Id between 1 and 15

续表

运　算　符	名　　称	示　　例	运　算　符	名　　称	示　　例
<	小于	id<5	IN	n/a	id in (3,4,5)
=>	大于等于	id=>5	NOT IN	n/a	name not in (shi,li)
<=	小于等于	id<=5	LIKE	模式匹配	name like ('shi%')
!=或<>	不等于	id!=5	NOT LIKE	模式匹配	name not like ('shi%')
IS NULL	n/a	id is null	REGEXP	常规表达式	name 正则表达式

表 9.1 中列举的是 WHERE 子句常用的比较运算符，例中的 id 是记录的编号，name 是表中的用户名。

例 9.3　应用 WHERE 子句查询 tb_login 表，条件是 user（用户名）为 mr，代码如下。**（实例位置：光盘\TM\sl\9\9.3）**

```
select * from tb_login where user = 'mr';
```

查询结果如图 9.2 所示。

```
mysql> select * from tb_login where user='mr';
+----+------+--------+---------+----------+
| id | user | pwd    | section | name     |
+----+------+--------+---------+----------+
|  8 | mr   | mrsoft | PHP     | 明日科技 |
+----+------+--------+---------+----------+
1 row in set (0.00 sec)

mysql>
```

图 9.2　查询指定数据

9.2.4　带关键字 IN 的查询

关键字 IN 可以判断某个字段的值是否在指定的集合中。如果字段的值在集合中，则满足查询条件，该记录将被查询出来；如果不在集合中，则不满足查询条件。其语法格式如下。

```
SELECT * FROM 表名 WHERE 条件 [NOT] IN(元素 1,元素 2,…,元素 n);
```

（1）[NOT]：是可选项，加上 NOT 表示不在集合内满足条件；

（2）元素：表示集合中的元素，各元素之间用逗号隔开，字符型元素需要加上单引号。

例 9.4　应用 IN 关键字查询 tb_login 表中 user 字段为 mr 和 lx 的记录，查询语句如下。**（实例位置：光盘\TM\sl\9\9.4）**

```
SELECT * FROM tb_login WHERE user IN('mr','lx');
```

查询结果如图 9.3 所示。

```
mysql> select * from tb_login where user IN('mr','lx');
+----+------+--------+---------------+----------+
| id | user | pwd    | section       | name     |
+----+------+--------+---------------+----------+
|  6 | lx   | lx     | PHP程序开发部 | 明日科技 |
|  8 | mr   | mrsoft | PHP           | 明日科技 |
+----+------+--------+---------------+----------+
2 rows in set (0.00 sec)

mysql>
```

图 9.3　使用关键字 IN 查询

例 9.5　使用关键字 NOT IN 查询 tb_login 表中 user 字段不为 mr 和 lx 的记录，查询语句如下。**（实例位置：光盘\TM\sl\9\9.5）**

```
SELECT * FROM tb_login WHERE user NOT IN('mr','lx');
```

查询结果如图 9.4 所示。

```
mysql> select * from tb_login where user NOT IN('mr','lx');
+----+------+------+----------+----------+
| id | user | pwd  | section  | name     |
+----+------+------+----------+----------+
|  7 | mrkj | mrkj | 程序开发 | 明日科技 |
+----+------+------+----------+----------+
1 row in set (0.00 sec)

mysql>
```

图 9.4　使用关键字 NOT IN 查询

9.2.5　带关键字 BETWEEN AND 的范围查询

关键字 BETWEEN AND 可以判断某个字段的值是否在指定的范围内。如果字段的值在指定范围内，则满足查询条件，该记录将被查询出来。如果不在指定范围内，则不满足查询条件。其语法如下。

```
SELECT * FROM 表名 WHERE 条件 [NOT] BETWEEN 取值 1 AND 取值 2;
```

（1）[NOT]：可选项，表示不在指定范围内满足条件。

（2）取值 1：表示范围的起始值。

（3）取值 2：表示范围的终止值。

例 9.6　查询 tb_login 表中 id 值在 5~7 之间的数据，查询语句如下。**（实例位置：光盘\TM\sl\9\9.6）**

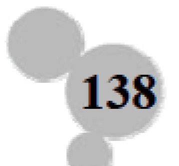
```
SELECT * FROM tb_login WHERE id BETWEEN 5 AND 7;
```

查询结果如图 9.5 所示。

```
mysql> select * from tb_login where id BETWEEN 5 and 7;
+----+------+------+---------------+----------+
| id | user | pwd  | section       | name     |
+----+------+------+---------------+----------+
|  6 | lx   | lx   | PHP程序开发部 | 明日科技 |
|  7 | mrkj | mrkj | 程序开发      | 明日科技 |
+----+------+------+---------------+----------+
2 rows in set (0.00 sec)

mysql>
```

图 9.5　使用关键字 BETWEEN AND 查询



如果要查询 tb_login 表中 id 值不在 5~7 之间的数据，则可以通过 NOT BETWEEN AND 来完成。其查询语句如下。

```
SELECT * FROM tb_login WHERE id NOT BETWEEN 5 AND 7;
```

9.2.6　带 LIKE 的字符匹配查询

LIKE 属于较常用的比较运算符，通过它可以实现模糊查询。它有两种通配符：“%”和下划线“_”。

（1）“%”可以匹配一个或多个字符，可以代表任意长度的字符串，长度可以为 0。例如，“明%技”表示以“明”开头，以“技”结尾的任意长度的字符串。该字符串可以代表“明日科技”“明日编程科技”“明日图书科技”等字符串。

（2）“_”只匹配一个字符。例如，m_n 表示以 m 开头，以 n 结尾的 3 个字符。中间的“_”可以代表任意一个字符。

说明

字符串“p”和“入”都算作一个字符，在这点上英文字母和中文是没有区别的。

例 9.7　查询 tb_login 表中 user 字段中包含 mr 字符的数据，查询语句如下。**（实例位置：光盘\TM\sl\9\9.7）**

```
select * from tb_login where user like '%mr%';
```

查询结果如图 9.6 所示。

```
mysql> select * from tb_login where user like '%mr%';
+----+------+--------+----------+----------+
| id | user | pwd    | section  | name     |
+----+------+--------+----------+----------+
|  7 | mrkj | mrkj   | 程序开发         | 明日科技         |
|  8 | mr   | mrsoft | PHP      | 明日科技     |
+----+------+--------+----------+----------+
2 rows in set (0.00 sec)
```

图 9.6　模糊查询

9.2.7　用关键字 IS NULL 查询空值

关键字 IS NULL 可以用来判断字段的值是否为空值（NULL）。如果字段的值是空值，则满足查询条件，该记录将被查询出来。如果字段的值不是空值，则不满足查询条件。其语法格式如下。

```
IS [NOT] NULL
```

其中，“NOT”是可选项，表示字段不是空值时满足条件。

例 9.8　下面使用关键字 IS NULL 查询 db_database09 数据库的 tb_book 表中 name 字段的值为空的记录，查询语句如下。**（实例位置：光盘\TM\sl\9\9.8）**

```
SELECT books,row FROM tb_book WHERE row IS NULL;
```

查询结果如图 9.7 所示。

```
mysql> select books,row from tb_book where row is null;
+------------------------+------+
| books                  | row  |
+------------------------+------+
| C#项目整合              | NULL |
| JAVA范例完全自学手册      | NULL |
+------------------------+------+
2 rows in set (0.00 sec)

mysql>
```

图 9.7　查询数据表 tb_book 中 row 字段值为空的记录

9.2.8　带关键字 AND 的多条件查询

关键字 AND 可以用来联合多个条件进行查询。使用关键字 AND 时，只有同时满足所有查询条件的记录会被查询出来。如果不满足这些查询条件的其中一个，这样的记录将被排除掉。关键字 AND 的语法格式如下。

```
select * from 数据表名 where 条件 1 and 条件 2 [···AND 条件表达式 n];
```

关键字 AND 连接两个条件表达式，可以同时使用多个关键字 AND 来连接多个条件表达式。

例 9.9　下面查询数据表 tb_login 中 user 字段值为 mr，并且 section 字段值为 PHP 的记录，查询语句如下。**（实例位置：光盘\TM\sl\9\9.9）**

```
select * from tb_login where user='mr' and section='php';
```

查询结果如图 9.8 所示。

```
mysql> select * from tb_login where user='mr' and section='php';
+----+------+--------+---------+----------+
| id | user | pwd    | section | name     |
+----+------+--------+---------+----------+
|  8 | mr   | mrsoft | PHP     | 明日科技   |
+----+------+--------+---------+----------+
1 row in set (0.01 sec)

mysql>
```

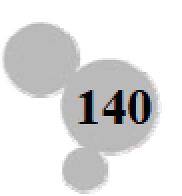
图 9.8　使用关键字 AND 实现多条件查询

9.2.9　带关键字 OR 的多条件查询

关键字 OR 也可以用来联合多个条件进行查询，但是与关键字 AND 不同，关键字 OR 只要满足查询条件中的一个，那么此记录就会被查询出来；如果不满足这些查询条件中的任何一个，这样的记录将被排除掉。关键字 OR 的语法格式如下。

```
select * from 数据表名 where 条件 1 OR 条件 2 [···OR 条件表达式 n];
```

关键字 OR 可以用来连接两个条件表达式。而且，可以同时使用多个关键字 OR 连接多个条件表达式。



例 9.10　下面查询 tb_login 表中 section 字段的值为“PHP”或者“程序开发”的记录，查询语句如下。**（实例位置：光盘\TM\sl\9\9.10）**

```
select * from tb_login where section='php' or section='程序开发';
```

查询结果如图 9.9 所示。

```
mysql> select * from tb_login where section='php' or section='程序开发';
+----+------+--------+----------+----------+
| id | user | pwd    | section  | name     |
+----+------+--------+----------+----------+
|  7 | mrkj | mrkj   | 程序开发 |          |
|  8 | mr   | mrsoft | PHP      | 明日科技 |
+----+------+--------+----------+----------+
2 rows in set (0.02 sec)
```

图 9.9　使用关键字 OR 实现多条件查询

9.2.10　用关键字 DISTINCT 去除结果中的重复行

使用关键字 DISTINCT 可以去除查询结果中的重复记录，语法格式如下。

```
select distinct 字段名 from 表名;
```

例 9.11　下面使用关键字 DISTINCT 去除 tb_login 表中 name 字段中的重复记录，查询语句如下。**（实例位置：光盘\TM\sl\9\9.11）**

```
select distinct name from tb_login;
```

查询结果如图 9.10 所示。去除重复记录前的 name 字段值如图 9.11 所示。

```
mysql> select distinct name from tb_login;
+----------+
| name     |
+----------+
| 高经理   |
| 明日科技 |
+----------+
2 rows in set (0.00 sec)

mysql>
```

图 9.10　使用关键字 DISTINCT 去除结果中的重复行



```
mysql> select name from tb_login;
+----------+
| name     |
+----------+
| 高经理   |
| 明日科技 |
| 明日科技 |
+----------+
3 rows in set (0.00 sec)

mysql> _
```

图 9.11　去除重复记录前的 name 字段值

9.2.11　用关键字 ORDER BY 对查询结果排序

使用关键字 ORDER BY 可以对查询的结果进行升序（ASC）和降序（DESC）排列，在默认情况下，ORDER BY 按升序输出结果。如果要按降序排列可以使用 DESC 来实现。语法格式如下。

```
ORDER BY 字段名 [ASC|DESC];
```

（1）ASC 表示按升序进行排序。

（2）DESC 表示按降序进行排序。

说明

对含有 NULL 值的列进行排序时，如果是按升序排列，NULL 值将出现在最前面，如果是按降序排列，NULL 值将出现在最后。

例 9.12 查询 tb_login 表中的所有信息，按照 id 序号进行降序排列，查询语句如下。（**实例位置：光盘\TM\sl\9\9.12**）

```
select * from tb_login order by id desc;
```

查询结果如图 9.12 所示。

```
mysql> select * from tb_login order by id desc;
+----+------+--------+--------------+--------+
| id | user | pwd    | section      | name   |
+----+------+--------+--------------+--------+
|  8 | mr   | mrsoft | PHP          | 明日科技 |
|  7 | mrkj | mrkj   | 程序开发       |        |
|  6 | lx   | lx     | PHP程序开发部        | 明日科技 |
+----+------+--------+--------------+--------+
3 rows in set (0.03 sec)
```

图 9.12　按 id 序号进行降序排列

9.2.12　用关键字 GROUP BY 分组查询

通过关键字 GROUP BY 可以将数据划分到不同的组中，实现对记录进行分组查询。在查询时，所查询的列必须包含在分组的列中，目的是使查询到的数据没有矛盾。

1．使用关键字 GROUP BY 来分组

单独使用关键字 GROUP BY 查询结果只显示每组的一条记录。

例 9.13 使用关键字 GROUP BY 对 tb_book 表中 talk 字段进行分组查询，查询语句如下。（**实例位置：光盘\TM\sl\9\9.13**）

```
select id,books,talk from tb_book GROUP BY talk;
```

查询结果如图 9.13 所示。

```
mysql> select id,books,talk from tb_book GROUP BY talk;
+----+---------------+-------+
| id | books         | talk  |
+----+---------------+-------+
| 27 | PHP典型模块          | .net  |
| 26 | JAVA典型模块         | JAVA  |
| 30 | aa            | PHP   |
| 29 | aaaa          | 其它     |
+----+---------------+-------+
4 rows in set (0.00 sec)

mysql>
```

图 9.13　使用关键字 GROUP BY 进行分组查询

为了使分组更加直观明了，下面查询数据表 tb_book 中的记录，查询结果如图 9.14 所示。



```
mysql> select id,books,talk,user from tb_book;
+----+-------------------------+-------+------+
| id | books                   | talk  | user |
+----+-------------------------+-------+------+
| 26 | JAVA典型模块                 | JAVA | mr   |
| 27 | PHP典型模块                  | .net | mr   |
| 28 | C#项目整合                   | .net | mr   |
| 29 | aaaa                    | 其它   | lx   |
| 30 | aa                      | PHP   | lx   |
| 25 | JAVA范例完全自学手册              | JAVA | mr   |
+----+-------------------------+-------+------+
6 rows in set (0.02 sec)

mysql> _
```

图 9.14　数据表 tb_book 中的记录

2．关键字 GROUP BY 与 GROUP_CONCAT()函数一起使用

使用关键字 GROUP BY 和 GROUP_CONCAT()函数查询，可以将每个组中的所有字段值都显示出来。

例 9.14　下面使用关键字 GROUP BY 和 GROUP_CONCAT()函数对 tb_book 表中的 talk 字段进行分组查询，查询语句如下。**（实例位置：光盘\TM\sl\9\9.14）**

```
select id,books,GROUP_CONCAT(talk) from tb_book GROUP BY talk;
```

查询结果如图 9.15 所示。

```
mysql> select id,books,GROUP_CONCAT(talk) from tb_book GROUP BY talk;
+----+--------------+--------------------+
| id | books        | GROUP_CONCAT(talk) |
+----+--------------+--------------------+
| 27 | PHP典型模块        | .net,.net          |
| 26 | JAVA典型模块       | JAVA,JAVA          |
| 30 | aa           | PHP                |
| 29 | aaaa         | 其它                 |
+----+--------------+--------------------+
4 rows in set (0.00 sec)

mysql>
```

图 9.15　使用关键字 GROUP BY 与 GROUP_CONCAT()函数进行分组查询

3．按多个字段进行分组

使用关键字 GROUP BY 也可以按多个字段进行分组。

例 9.15　下面对数据表 tb_book 表中的 user 字段和 sort 字段进行分组，分组过程中，先按照 talk 字段进行分组。当 talk 字段的值相等时，再按照 sort 字段进行分组，查询语句如下。**（实例位置：光盘\TM\sl\9\9.15）**

```
select id,books,talk,user from tb_book GROUP BY user,talk;
```

查询结果如图 9.16 所示。

```
mysql> select id,books,talk,user from tb_book GROUP BY user,talk;
+----+--------------+------+------+
| id | books        | talk | user |
+----+--------------+------+------+
| 30 | aa           | PHP  | lx   |
| 29 | aaaa         | 其它  | lx   |
| 27 | PHP典型模块        | .net | mr   |
| 26 | JAVA典型模块       | JAVA | mr   |
+----+--------------+------+------+
4 rows in set (0.00 sec)
```

图 9.16　使用关键字 GROUP BY 实现多个字段分组

9.2.13　用关键字 LIMIT 限制查询结果的数量

查询数据时，可能会查询出很多的记录，而用户需要的记录可能只是很少的一部分，这样就需要来限制查询结果的数量。LIMIT 是 MySQL 中的一个特殊关键字。关键字 LIMIT 可以对查询结果的记录条数进行限定，控制它输出的行数。下面通过具体实例来了解关键字 LIMIT 的使用方法。

例 9.16　查询数据表 tb_login 中，按照 id 编号进行升序排列，显示前 3 条记录，查询语句如下。**（实例位置：光盘\TM\sl\9\9.16）**

```
select * from tb_login order by id asc limit 3;
```

查询结果如图 9.17 所示。

```
mysql> select * from tb_login order by id asc limit 3;
+----+------+--------+-----------------+-----------+
| id | user | pwd    | section         | name      |
+----+------+--------+-----------------+-----------+
|  6 | lx   | lx     | PHP程序开发部   | 明日科技  |
|  7 | mrkj | mrkj   | 程序开发        |           |
|  8 | mr   | mrsoft | PHP             | 明日科技  |
+----+------+--------+-----------------+-----------+
3 rows in set (0.00 sec)
```

图 9.17　使用关键字 LIMIT 查询指定记录数

使用关键字 LIMIT 还可以从查询结果的中间部分取值。首先要定义两个参数，参数 1 是开始读取的第一条记录的编号（在查询结果中，第一个结果的记录编号是 0，而不是 1）；参数 2 是要查询记录的个数。

例 9.17　查询 tb_login 表中，按照 id 编号进行升序排列，从编号 1 开始，查询两条记录，查询语句如下。**（实例位置：光盘\TM\sl\9\9.17）**

```
select * from tb_login where id order by id asc limit 1,2;
```

查询结果如图 9.18 所示。

```
mysql> select * from tb_login order by id asc limit 1,2;
+----+------+--------+-----------+----------+
| id | user | pwd    | section   | name     |
+----+------+--------+-----------+----------+
|  7 | mrkj | mrkj   | 程序开发  | 高经理   |
|  8 | mr   | mrsoft | PHP       | 明日科技 |
+----+------+--------+-----------+----------+
2 rows in set (0.00 sec)

mysql>
```

图 9.18　使用关键字 LIMIT 查询指定记录

9.3　聚合函数查询

聚合函数的最大特点是它们根据一组数据求出一个值。聚合函数的结果值只根据选定行中非



NULL 的值进行计算，NULL 值被忽略。

9.3.1　COUNT()函数

COUNT()函数，对于除“*”以外的任何参数，返回所选择集合中非 NULL 值的行的数目；对于参数“*”，返回选择集合中所有行的数目，包含 NULL 值的行。没有 WHERE 子句的 COUNT(*)是经过内部优化的，能够快速返回表中所有的记录总数。

例 9.18　下面使用 count()函数统计数据表 tb_login 中的记录数，查询语句如下。**（实例位置：光盘\TM\sl\9\9.18）**

```
select count(*) from tb_login;
```

查询结果如图 9.19 所示。结果显示，数据表 tb_login 中共有 4 条记录。

```
mysql> select count(*) from tb_login;
+----------+
| count(*) |
+----------+
|        4 |
+----------+
1 row in set (0.00 sec)

mysql> _
```

图 9.19　使用 count()函数统计记录数

9.3.2　SUM()函数

SUM()函数可以求出表中某个字段取值的总和。

例 9.19　使用 SUM()函数统计数据表 tb_book 中图书的访问量字段（row）的总和。在查询前，先来查询一下 tb_book 表中 row 字段的值，结果如图 9.20 所示。**（实例位置：光盘\TM\sl\9\9.19）**

```
mysql> select row from tb_book;
+------+
| row  |
+------+
| 12   |
| 95   |
| NULL |
| 1    |
| 8    |
| NULL |
+------+
6 rows in set (0.00 sec)
```

图 9.20　tb_book 表中 row 字段的值

下面使用 SUM()函数来查询，查询语句如下。

```
select sum(row) from tb_book;
```

查询结果如图 9.21 所示，显示 row 字段的总和为 116。

```
mysql> select sum(row) from tb_book;
+----------+
| sum(row) |
+----------+
|      116 |
+----------+
1 row in set (0.02 sec)

mysql> _
```

图 9.21　使用 SUM()函数查询 row 字段值的总和

9.3.3　AVG()函数

AVG()函数可以求出表中某个字段取值的平均值。

例 9.20　下面使用 AVG()函数求数据表 tb_book 中 row 字段值的平均值，查询语句如下。**（实例位置：光盘\TM\sl\9\9.20）**

```
select AVG(row) from tb_book;
```

查询结果如图 9.22 所示。

```
mysql> select AVG(row) from tb_book;
+----------+
| AVG(row) |
+----------+
|       29 |
+----------+
1 row in set (0.00 sec)

mysql> _
```

图 9.22　使用 AVG()函数求 row 字段值的平均值

9.3.4　MAX()函数

MAX()函数可以求出表中某个字段取值的最大值。

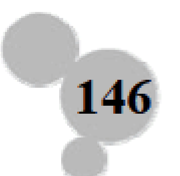
例 9.21　下面使用 MAX()函数查询数据表 tb_book 中 row 字段的最大值，查询语句如下。**（实例位置：光盘\TM\sl\9\9.21）**

```
select MAX(row) from tb_book;
```

查询结果如图 9.23 所示。

```
mysql> select MAX(row) from tb_book;
+----------+
| MAX(row) |
+----------+
| 95       |
+----------+
1 row in set (0.00 sec)
```

图 9.23　使用 MAX()函数求 row 字段的最大值



下面来看一下数据表 tb_book 中 row 字段的所有值，查询结果如图 9.24 所示。结果显示 row 字段中最大值为 95，与使用 MAX 函数查询的结果一致。

```
mysql> select row from tb_book;
+------+
| row  |
+------+
| 12   |
| 95   |
| NULL |
| 1    |
| 8    |
| NULL |
+------+
6 rows in set (0.00 sec)

mysql>
```

图 9.24　数据表 tb_book 中 row 字段的所有值

9.3.5　MIN()函数

MIN()函数可以求出表中某个字段取值的最小值。

例 9.22　使用 MIN()函数查询数据表 tb_book 中 row 字段的最小值，查询语句如下。（**实例位置：光盘\TM\sl\9\9.22**）

```
select MIN(row) from tb_book;
```

查询结果如图 9.25 所示。

```
mysql> select MIN(row) from tb_book;
+----------+
| MIN(row) |
+----------+
| 1        |
+----------+
1 row in set (0.00 sec)
```

图 9.25　使用 MIN()函数求 row 字段的最小值

9.4　连 接 查 询

连接是把不同表的记录连到一起的最普遍的方法。一种错误的观念认为由于 MySQL 的简单性和源代码开放性，使它不擅长连接。MySQL 从一开始就能够很好地支持连接，现在还以支持标准的 SQL92 连接语句而自豪，这种连接语句可以以多种高级方法来组合表记录。

9.4.1　内连接查询

内连接是最普遍的连接类型，而且是最匀称的，因为它们要求构成连接的每一部分的每个表的匹配，不匹配的行将被排除。

内连接的最常见的例子是相等连接，也就是连接后的表中的某个字段与每个表中的都相同。这种情况下，最后的结果集只包含参加连接的表中与指定字段相符的行。

例 9.23　下面有两个表，tb_login 用户信息表和 tb_book 图书信息表，先来分别看下各表的数据，图 9.26 为 tb_login 表的数据，图 9.27 为 tb_book 表的数据。**（实例位置：光盘\TM\sl\9\9.23）**

```
mysql> select * from tb_login;
+----+------+--------+-----------------+----------+
| id | user | pwd    | section         | name     |
+----+------+--------+-----------------+----------+
|  7 | mrkj | mrkj   | 程序开发            |          |
|  6 | lx   | lx     | PHP程序开发部             | 明日科技       |
|  8 | mr   | mrsoft | PHP             | 明日科技       |
+----+------+--------+-----------------+----------+
3 rows in set (0.00 sec)
```

图 9.26　数据表 tb_login

```
mysql> select id,user,sort,books from tb_book;
+----+------+-------------+-----------------------+
| id | user | sort        | books                 |
+----+------+-------------+-----------------------+
| 26 | mr   | 典型应用类            | JAVA典型模块          |
| 27 | mr   | 模块类          | PHP典型模块           |
| 28 | mr   | 项目类          | C#项目整合            |
| 29 | lx   | 范例类          | aaaa                  |
| 30 | lx   | 基础类          | aa                    |
| 25 | mr   | 范例类          | JAVA范例完全自学手册  |
+----+------+-------------+-----------------------+
6 rows in set (0.00 sec)
```

图 9.27　数据表 tb_book

从上面的查询结果中可以看出，在两个表中存在一个连接——user 字段，它在两个表中是等同的，tb_login 表的 user 字段与 tb_book 表的 user 字段相等，因此可以创建两个表的一个连接。查询语句如下。

```
select name,books from tb_login,tb_book where tb_login.user=tb_book.user;
```

查询结果如图 9.28 所示。

```
mysql> select name,books from tb_login,tb_book where tb_login.user=tb_book.user;

+----------+-----------------------+
| name     | books                 |
+----------+-----------------------+
| 明日科技         | JAVA典型模块          |
| 明日科技         | PHP典型模块           |
| 明日科技         | C#项目整合            |
| 明日科技         | Word范例宝典          |
| 明日科技         | PHP24堂课             |
| 高经理         | JAVA范例完全自学手册        |
| 明日科技         | VB典型模块            |
| 高经理         | PHP范例完全自学手册         |
| 明日科技         | C范例宝典             |
| 明日科技         | PHP项目整合           |
| 明日科技         | PHP24堂课             |
| 明日科技         | VB范例宝典            |
| 明日科技         | PHP网络编程自学手册        |
| 明日科技         | PHP编程宝典           |
+----------+-----------------------+
14 rows in set (0.00 sec)

mysql>
```

图 9.28　内连接查询

9.4.2　外连接查询

与内连接不同，外连接是指使用 OUTER JOIN 关键字将两个表连接起来。外连接生成的结果集不仅包含符合连接条件的行数据，而且还包括左表（左外连接时的表）、右表（右外连接时的表）或两边连接表（全外连接时的表）中所有的数据行。语法格式如下。

```
SELECT 字段名称 FROM 表名 1 LEFT|RIGHT JOIN 表名 2 ON 表名 1.字段名 1=表名 2.属性名 2;
```

外连接分为左外连接（LEFT JOIN）、右外连接（RIGHT JOIN）和全外连接 3 种类型。

1. 左外连接

左外连接（LEFT JOIN）是指将左表中的所有数据分别与右表中的每条数据进行连接组合，返回的结果除内连接的数据外，还包括左表中不符合条件的数据，并在右表的相应列中添加 NULL 值。

例 9.24　下面使用左外连接查询 tb_login 表和 tb_book 表，通过 user 字段进行连接，查询语句如下。**（实例位置：光盘\TM\sl\9\9.24）**

```
select section,tb_login.user,books,row from tb_login left join tb_book on tb_login.user=tb_book.user;
```

查询结果如图 9.29 所示。

```
mysql> select section,tb_login.user,books,row
    -> from tb_login
    -> left join tb_book
    -> on tb_login.user=tb_book.user;
+----------------+------+-----------------------------+------+
| section        | user | books                       | row  |
+----------------+------+-----------------------------+------+
| 程序开发        | mrkj | NULL                        | NULL |
| PHP程序开发部   | lx   | aaaa                        | 1    |
| PHP程序开发部   | lx   | aa                          | 8    |
| PHP            | mr   | JAVA典型模块                 | 12   |
| PHP            | mr   | PHP典型模块                  | 95   |
| PHP            | mr   | C#项目整合                   | NULL |
| PHP            | mr   | JAVA范例完全自学手册          | NULL |
+----------------+------+-----------------------------+------+
7 rows in set (0.00 sec)
```

图 9.29　左外连接查询

结果显示，第一条记录的 books 和 row 字段的值为空，这是因为在 tb_book 表中并不存在 user 字段为 mrkj 的值。

2. 右外连接

右外连接（RIGHT JOIN）是指将右表中的所有数据分别与左表中的每条数据进行连接组合，返回的结果除内连接的数据外，还包括右表中不符合条件的数据，并在左表的相应列中添加 NULL。

例 9.25　下面使用右外连接查询 tb_book 表和 tb_login 表，两表通过 user 字段连接，查询语句如下。**（实例位置：光盘\TM\sl\9\9.25）**

```
select section,tb_book.user,books,row from tb_book right join tb_login on tb_book.user=tb_login.user;
```

查询结果如图 9.30 所示。

```
mysql> select section,tb_book.user,books,row from tb_book right join tb_login on
 tb_book.user=tb_login.user;
+----------------+------+------------------------+------+
| section        | user | books                  | row  |
+----------------+------+------------------------+------+
| 程序开发       | NULL | NULL                   | NULL |
| PHP程序开发部  | lx   | aaaa                   | 1    |
| PHP程序开发部  | lx   | aa                     | 8    |
| PHP            | mr   | JAVA典型模块           | 12   |
| PHP            | mr   | PHP典型模块            | 95   |
| PHP            | mr   | C#项目整合             | NULL |
| PHP            | mr   | JAVA范例完全自学手册   | NULL |
+----------------+------+------------------------+------+
7 rows in set (0.00 sec)

mysql>
```

图 9.30　右外连接查询

9.4.3　复合条件连接查询

在连接查询时，也可以增加其他的限制条件。通过多个条件的复合查询，可以使查询结果更加准确。

例 9.26　下面使用内连接查询 tb_book 表和 tb_login 表，并且 tb_book 表中 row 字段值必须大于 5，查询语句如下。**（实例位置：光盘\TM\sl\9\9.26）**

```
select section,tb_book.user,books,row from tb_book,tb_login where tb_book.user=tb_login.user and row>5;
```

查询结果如图 9.31 所示。

```
mysql> select section,tb_book.user,books,row from tb_book,tb_login where tb_book
.user=tb_login.user and row>5;
+----------------+------+--------------+------+
| section        | user | books        | row  |
+----------------+------+--------------+------+
| PHP            | mr   | JAVA典型模块 | 12   |
| PHP            | mr   | PHP典型模块  | 95   |
| PHP程序开发部  | lx   | aa           | 8    |
+----------------+------+--------------+------+
3 rows in set (0.03 sec)

mysql>
```

图 9.31　复合条件连接查询

9.5　子　查　询

子查询就是 SELECT 查询是另一个查询的附属。MySQL 4.1 可以嵌套多个查询，在外面一层的查询中使用里面一层查询产生的结果集。这样就不是执行两个（或者多个）独立的查询，而是执行包含一个（或者多个）子查询的单独查询。

当遇到这样的多层查询时，MySQL 从最内层的查询开始，然后从它开始向外向上移动到外层（主）查询，在这个过程中每个查询产生的结果集都被赋给包围它的父查询，接着这个父查询被执行，它的结果也被指定给父查询。

除了结果集经常由包含一个或多个值的一列组成外，子查询和常规 SELECT 查询的执行方式一样。子查询可以用在任何可以使用表达式的地方，它必须由父查询包围，而且，如同常规的 SELECT 查询，

它必须包含一个字段列表（这是一个单列列表）、一个具有一个或者多个表名字的 FROM 子句以及可选的 WHERE、HAVING 和 GROUP BY 子句。

9.5.1　带关键字 IN 的子查询

只有子查询返回的结果列包含一个值时，比较运算符才适用。假如一个子查询返回的结果集是值的列表，这时比较运算符就必须用关键字 IN 代替。

IN 运算符可以检测结果集中是否存在某个特定的值，如果检测成功就执行外部的查询。

例 9.27　下面查询 tb_login 表中的记录，但 user 字段值必须在 tb_book 表中的 user 字段中出现过，查询语句如下。**（实例位置：光盘\TM\sl\9\9.27）**

```
select * from tb_login where user in(select user from tb_book);
```

在查询前，先来分别看一下 tb_login 和 tb_book 表中的 user 字段值，以便进行对比，tb_login 表中的 user 字段值如图 9.32 所示。tb_book 表中的 user 字段值如图 9.33 所示。

```
mysql> select user from tb_login;
+-------+
| user |
+-------+
| mrkj |
| lx   |
| mr   |
+-------+
3 rows in set (0.00 sec)

mysql> _
```

图 9.32　tb_login 表中的 user 字段值



```
mysql> select user from tb_book;
+-------+
| user |
+-------+
| mr   |
| mr   |
| mr   |
| lx   |
| lx   |
| mr   |
| lx   |
| mr   |
| lx   |
| lx   |
| mr   |
| mr   |
| lx   |
| mr   |
+-------+
14 rows in set (0.00 sec)

mysql>
```

图 9.33　tb_book 表中的 user 字段值

从上面的查询结果可以看出，在 tb_book 表的 user 字段中没有出现 mrkj 值。下面执行带关键字 IN 的子查询语句，查询结果如图 9.34 所示。

```
mysql> select * from tb_login where user in(select user from tb_book);
+----+------+--------+---------------+----------+
| id | user | pwd    | section       | name     |
+----+------+--------+---------------+----------+
|  6 | lx   | lx     | PHP程序开发部 | 明日科技 |
|  8 | mr   | mrsoft | PHP           | 明日科技 |
+----+------+--------+---------------+----------+
2 rows in set (0.00 sec)
```

图 9.34　使用关键字 IN 实现子查询

查询结果只查询出了 user 字段值为 lx 和 mr 的记录，因为在 tb_book 表的 user 字段中没有出现 mrkj 的值。


说明

关键字 NOT IN 的作用与关键字 IN 刚好相反。在本例中，如果将 IN 换为 NOT IN，则查询结果将会只显示一条 user 字段值为 mrkj 的记录。

9.5.2 带比较运算符的子查询

子查询可以使用比较运算符，包括=、!=、>、>=、<和<=等。比较运算符在子查询时使用非常广泛。

例 9.28 下面查询图书访问量为“优秀”的图书，在 tb_row 表中将图书访问量按访问数划分等级，如图 9.35 所示。**（实例位置：光盘\TM\sl\9\9.28）**

从结果中看出，当访问量大于等于 90 时即为“优秀”，下面再来查询 tb_book 图书信息表中 row 字段的值，如图 9.36 所示。

```
mysql> select * from tb_row;
+----+-----+------+
| id | row | name |
+----+-----+------+
|  1 | 90  | 优秀 |
|  2 | 80  | 良好 |
|  3 | 70  | 一般 |
|  4 | 50  | 差   |
+----+-----+------+
4 rows in set (0.00 sec)
```

图 9.35 查询 tb_row 表中的数据



```
mysql> select id,books,row from tb_book;
+----+----------------------+-----+
| id | books                | row |
+----+----------------------+-----+
| 26 | JAVA典型模块          | 12  |
| 27 | PHP典型模块           | 95  |
| 28 | C#项目整合            |     |
| 29 | Word范例宝典          | 1   |
| 30 | PHP24堂课             | 8   |
| 25 | JAVA范例完全自学手册   |     |
| 24 | VB典型模块            | 1   |
| 14 | PHP范例完全自学手册    |     |
| 20 | C范例宝典             |     |
| 21 | PHP项目整合           |     |
| 22 | PHP24堂课             |     |
| 23 | VB范例宝典            | 1   |
| 31 | PHP网络编程自学手册     | 12  |
| 34 | PHP编程宝典           | 12  |
+----+----------------------+-----+
14 rows in set (0.00 sec)

mysql> _
```

图 9.36 查询 tb_book 表中 row 字段的值

结果显示，第 27 条记录的访问量大于 90。下面使用比较运算符的子查询方式来查询访问量为优秀的图书信息，查询语句如下。

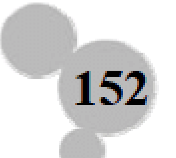
```
select id,books,row from tb_book where row>=(select row from tb_row where id=1);
```

查询结果如图 9.37 所示。

```
mysql> select id,books,row from tb_book where row>=(select row from tb_row where
 id=1);
+----+--------------+-----+
| id | books        | row |
+----+--------------+-----+
| 27 | PHP典型模块   | 95  |
+----+--------------+-----+
1 row in set (0.00 sec)

mysql> _
```

图 9.37 使用比较运算符的子查询方式来查询访问量为“优秀”的图书信息



9.5.3　带关键字 EXISTS 的子查询

使用关键字 EXISTS 时，内层查询语句不返回查询的记录。而是返回一个真假值。如果内层查询语句查询到满足条件的记录，就返回一个真值（true），否则将返回一个假值（false）。当返回的值为 true 时，外层查询语句将进行查询；当返回的值为 false 时，外层查询语句不进行查询或者查询不出任何记录。

例 9.29　下面使用子查询查询 tb_book 表中是否存在 id 值为 27 的记录，如果存在则查询 tb_row 表中的记录，如果不存在则不执行外层查询，查询语句如下。**（实例位置：光盘\TM\sl\9\9.29）**

```
select * from tb_row where exists (select * from tb_book where id=27);
```

查询结果如图 9.38 所示。

```
mysql> select * from tb_row where exists(select * from tb_book where id=27);
+----+-----+------+
| id | row | name |
+----+-----+------+
|  1 | 90  | 优秀     |
|  2 | 80  | 良好     |
|  3 | 70  | 一般    |
|  4 | 50  | 差     |
+----+-----+------+
4 rows in set (0.00 sec)

mysql> _
```

图 9.38　使用关键字 EXISTS 的子查询

因为子查询 tb_book 表中存在 id 值为 27 的记录，即返回值为真，外层查询接收到真值后，开始执行查询。

当关键字 EXISTS 与其他查询条件一起使用时，需要使用 AND 或者 OR 来连接表达式与 EXISTS 关键字。

例 9.30　如果 tb_row 表中存在 name 值为“优秀”的记录，则查询 tb_book 表中 row 字段大于等于 90 的记录，查询语句如下。**（实例位置：光盘\TM\sl\9\9.30）**

```
select id,books,row from tb_book where row>=90 and exists(select * from tb_row where name='优秀');
```

查询结果如图 9.39 所示。

```
mysql> select id,books,row from tb_book where row>=90 and exists(select * from t
b_row where name='优秀');
+----+--------------+-----+
| id | books        | row |
+----+--------------+-----+
| 27 | PHP典型模块         | 95  |
+----+--------------+-----+
1 row in set (0.02 sec)

mysql> _
```

图 9.39　使用关键字 EXISTS 查询 tb_book 表中 row 字段大于等于 90 的记录



说明

与关键字 EXISTS 刚好相反，使用关键字 NOT EXISTS 时，当返回的值是 true 时，外层查询语句不执行查询；当返回值是 false 时，外层查询语句将执行查询。

9.5.4 带关键字 ANY 的子查询

关键字 ANY 表示满足其中任意一个条件。使用关键字 ANY 时，只要满足内层查询语句返回的结果中的任意一个，就可以通过该条件来执行外层查询语句。

例 9.31 查询 tb_book 表中 row 字段的值小于 tb_row 表中 row 字段最小值的记录，首先查询出 tb_row 表中 row 字段的值，然后使用关键字 ANY（“<ANY”表示小于所有值）判断，查询语句如下。**（实例位置：光盘\TM\sl\9\9.31）**

```
select books,row from tb_book where row<ANY(select row from tb_row);
```

查询结果如图 9.40 所示。

```
mysql> select books,row from tb_book where row<ANY(select row from tb_row);
+---------------+------+
| books         | row  |
+---------------+------+
| JAVA典型模块        | 12   |
| aaaa          | 1    |
| aa            | 8    |
+---------------+------+
3 rows in set (0.00 sec)

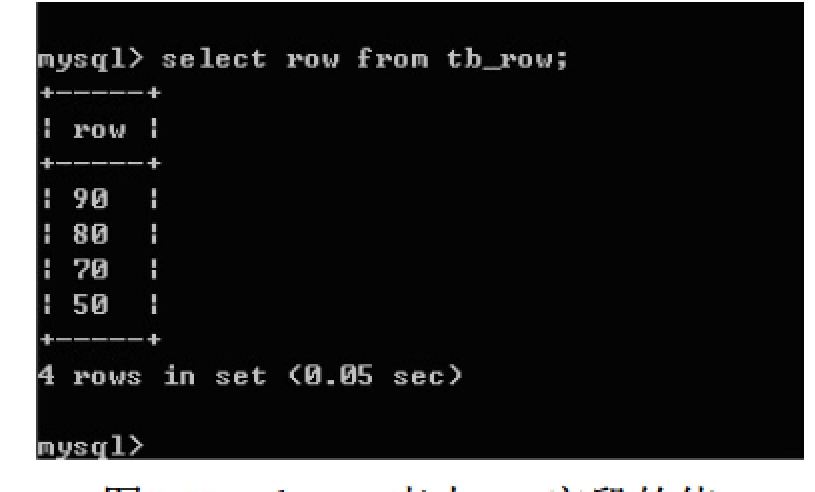
mysql>
```

图 9.40 使用关键字 ANY 实现子查询

为了使结果更加直观，下面分别查询 tb_book 表和 tb_row 表中的 row 字段值，查询结果如图 9.41 和图 9.42 所示。

```
mysql> select row from tb_book;
+------+
| row  |
+------+
| 12   |
| 95   |
| NULL |
| 1    |
| 8    |
| NULL |
+------+
6 rows in set (0.06 sec)
```

图9.41 tb_book表中row字段的值



```
mysql> select row from tb_row;
+-----+
| row |
+-----+
| 90  |
| 80  |
| 70  |
| 50  |
+-----+
4 rows in set (0.05 sec)

mysql>
```

图9.42 tb_row表中row字段的值

结果显示，tb_row 表中 row 字段的最小值为 50，在 tb_book 表中 row 字段小于 50 的记录有 3 条，与带关键字 ANY 的子查询结果相同。

9.5.5 带关键字 ALL 的子查询

关键字 ALL 表示满足所有条件。使用关键字 ALL 时，只有满足内层查询语句返回的所有结果，



才可以执行外层查询语句。

例 9.32　查询 tb_book 表中 row 字段的值大于 tb_row 表中 row 字段最大值的记录，首先使用子查询，查询出 tb_row 表中 row 字段的值，然后使用 ALL 关键字（“>=ALL”表示大于等于所有值）判断，查询语句如下。**（实例位置：光盘\TM\sl\9\9.32）**

```
select books,row from tb_book where row>=ALL(select row from tb_row);
```

查询结果如图 9.43 所示。

```
mysql> select books,row from tb_book where row>=all(select row from tb_row);
+--------------+------+
| books        | row  |
+--------------+------+
| PHP典型模块         | 95   |
+--------------+------+
1 row in set (0.00 sec)

mysql>
```

图 9.43　使用关键字 ALL 实现子查询

为了使结果更加直观，下面分别查询 tb_book 表和 tb_row 表中的 row 字段值，查询结果如图 9.44 和图 9.45 所示。

```
mysql> select row from tb_book;
+------+
| row  |
+------+
| 12   |
| 95   |
| NULL |
| 1    |
| 8    |
| NULL |
+------+
6 rows in set (0.06 sec)
```

图 9.44　tb_book 表中 row 字段的值

```
mysql> select row from tb_row;
+-----+
| row |
+-----+
| 90  |
| 80  |
| 70  |
| 50  |
+-----+
4 rows in set (0.05 sec)

mysql>
```

图 9.45　tb_row 表中 row 字段的值

结果显示，tb_row 表中 row 字段的最大值为 90，在 tb_book 表中 row 字段大于 90 的只有第 2 条记录 95，与带关键字 ALL 的子查询结果相同。

说明

关键字 ANY 和关键字 ALL 的使用方式是一样的，但是这两者有很大的区别。使用关键字 ANY 时，只要满足内层查询语句返回的结果中的任何一个，就可以通过该条件来执行外层查询语句；而关键字 ALL 则需要满足内层查询语句返回的所有结果，才可以执行外层查询语句。

9.6　合并查询结果

合并查询结果是将多个 SELECT 语句的查询结果合并到一起。因为某种情况下，需要将几个 SELECT 语句查询出来的结果合并起来显示。合并查询结果使用关键字 UNION 和 UNION ALL。关键

字 UNION 是将所有的查询结果合并到一起，然后去除相同记录；而关键字 UNION ALL 则只是简单地将结果合并到一起，下面分别介绍这两种合并方法。

1. UNION

例 9.33 下面查询 tb_book 表和 tb_login 表中的 user 字段，并使用 UNION 关键字合并查询结果。在执行查询操作前，先来看一下 tb_book 表和 tb_login 表中 user 字段的值，查询结果如图 9.46 和图 9.47 所示。**（实例位置：光盘\TM\sl\9\9.33）**

```
mysql> select user from tb_book;
+------+
| user |
+------+
| mr   |
| mr   |
| mr   |
| lx   |
| lx   |
| mr   |
+------+
6 rows in set (0.06 sec)
```

图 9.46 tb_book 表中 user 字段的值

```
mysql> select user from tb_login;
+------+
| user |
+------+
| mrkj |
| lx   |
| mr   |
+------+
3 rows in set (0.03 sec)

mysql> _
```

图 9.47 tb_login 表中 user 字段的值

结果显示，在 tb_book 表中 user 字段的值有两种，分别为 mr 和 lx，而 tb_login 表中 user 字段的值有三种。下面使用关键字 UNION 合并两个表的查询结果，查询语句如下。

```
select user from tb_book
UNION
select user from tb_login;
```

查询结果如图 9.48 所示。结果显示，合并后将所有结果合并到了一起，并去除了重复值。

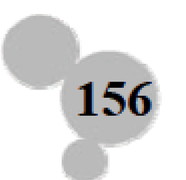
```
mysql> select user from tb_book
    -> union
    -> select user from tb_login;
+------+
| user |
+------+
| mr   |
| lx   |
| mrkj |
+------+
3 rows in set (0.00 sec)
```

图 9.48 使用关键字 UNION 合并查询结果

2. UNION ALL

查询 tb_book 表和 tb_login 表中的 user 字段，并使用关键字 UNION ALL 合并查询结果，查询语句如下。

```
select user from tb_book
UNION ALL
```



```
select user from tb_login;
```

查询结果如图 9.49 所示。tb_book 表和 tb_login 表的记录请参见例 9.33。

```
mysql> select user from tb_book
    -> union all
    -> select user from tb_login;
+-------+
| user  |
+-------+
| mr    |
| mr    |
| mr    |
| lx    |
| lx    |
| mr    |
| mrkj  |
| lx    |
| mr    |
+-------+
9 rows in set (0.00 sec)

mysql>
```

图 9.49 使用关键字 UNION ALL 合并查询结果

9.7 定义表和字段的别名

在查询时，可以为表和字段取一个别名，这个别名可以代替其指定的表和字段。为字段和表取别名，能够使查询更加方便，而且可以使查询结果以更加合理的方式显示。

9.7.1 为表取别名

当表的名称特别长时，在查询中直接使用表名很不方便，这时可以为表取一个贴切的别名。

例 9.34 下面为 tb_program 表取别名为 p，然后查询 tb_program 表中 talk 字段值为 php 的记录，查询语句如下。**（实例位置：光盘\TM\sl\9\9.34）**

```
select * from tb_program p where p.talk='PHP';
```

tb_program p 表示 tb_program 表的别名为 p；p.talk 表示 tb_program 表中的 talk 字段。查询结果如图 9.50 所示。

```
mysql> select * from tb_program p where p.talk='PHP';
+----+------+
| id | talk |
+----+------+
|  1 | PHP  |
+----+------+
1 row in set (0.00 sec)

mysql>
```

图 9.50 为表取别名

9.7.2 为字段取别名

当查询数据时，MySQL 会显示每个输出列的名词。默认情况下，显示的列名是创建表时定义的列名。同样可以为这个列取一个别名。

MySQL 中为字段取别名的基本形式如下。

```
字段名 [AS] 别名
```

例 9.35 下面为 tb_login 表中的 section 和 name 字段分别取别名为 login_section 和 login_name，SQL 代码如下。（**实例位置：光盘\TM\sl\9\9.35**）

```
select section AS login_section,name AS login_name from tb_login;
```

查询结果如图 9.51 所示。

```
mysql> select section AS login_section,name AS login_name from tb_login;
+---------------+------------+
| login_section | login_name |
+---------------+------------+
| 程序开发              | 高经理          |
| PHP程序开发部            | 明日科技          |
| PHP           | 明日科技          |
+---------------+------------+
3 rows in set (0.03 sec)

mysql>
```

图 9.51 为字段取别名

9.8 使用正则表达式查询

正则表达式是用某种模式去匹配一类字符串的一个方式。正则表达式的查询能力比通配字符的查询能力更强大，而且更加灵活。下面详细讲解如何使用正则表达式来查询。

在 MySQL 中，使用关键字 REGEXP 来匹配查询正则表达式，其基本形式如下。

```
字段名 REGEXP '匹配方式'
```

（1）字段名：表示需要查询的字段名称。

（2）匹配方式：表示以哪种方式来进行匹配查询。其支持的模式匹配字符如表 9.2 所示。

表 9.2 正则表达式的模式字符

模式字符	含义	应用举例
^	匹配以特定字符或字符串开头的记录	使用“^”表达式查询 tb_book 表中 books 字段以字母 php 开头的记录，语句如下： select books from tb_book where books REGEXP '^php';

续表

模 式 字 符	含　义	应 用 举 例
$	匹配以特定字符或字符串结尾的记录	使用“$”表达式查询 tb_book 表中 books 字段以“模块”结尾的记录，语句如下： select books from tb_book where books REGEXP '模块$';
.	匹配字符串的任意一个字符，包括回车和换行符	使用“.”表达式来查询 tb_book 表中 books 字段中包含 P 字符的记录，语句如下： select books from tb_book where books REGEXP 'P.';
[字符集合]	匹配“字符集合”中的任意一个字符	使用“[]”表达式来查询 tb_book 表中 books 字段中包含 PCA 字符的记录，语句如下： select books from tb_book where books REGEXP '[PCA]';
[^字符集合]	匹配除“字符集合”以外的任意一个字符	查询 tb_program 表中 talk 字段值中包含 c~z 字母以外的记录，语句如下： select talk from tb_program where talk regexp '[^c-z]';
S1|S2|S3	匹配 S1、S2 和 S3 中的任意一个字符串	查询 tb_books 表中 books 字段中包含 php、c 或者 java 字符中任意一个字符的记录，语句如下： select books from tb_books where books regexp 'php|c|java';
*	匹配多个该符号之前的字符，包括 0 和 1 个	使用“*”表达式查询 tb_book 表中 books 字段中 A 字符前出现过 J 字符的记录，语句如下： select books from tb_book where books regexp 'J*A';
+	匹配多个该符号之前的字符，包括 1 个	使用“+”表达式来查询 tb_book 表中 books 字段中 A 字符前面至少出现过一个 J 字符，语句如下： select books from tb_book where books regexp 'J+A';
字符串{N}	匹配字符串出现 N 次	使用{N}表达式查询 tb_book 表中 books 字段中连续出现 3 次 a 字符的记录，语句如下： select books from tb_book where books regexp 'a{3}';
字符串{M,N}	匹配字符串出现至少 M 次，最多 N 次	使用{M,N}表达式查询 tb_book 表中 books 字段中最少出现两次，最多出现 4 次 a 字符的记录，语句如下： select books from tb_book where books regexp 'a{2,4}';

这里的正则表达式与 Java 语言、PHP 语言等编程语言中的正则表达式基本一致。

9.8.1　匹配指定字符中的任意一个

使用方括号（[]）可以将需要查询字符组成一个字符集。只要记录中包含方括号中的任意字符，该记录将会被查询出来。例如，通过“[abc]”可以查询包含 a、b 和 c3 个字母中任何一个的记录。

例 9.36　下面从 info 表 name 字段中查询包含 c、e 和 o 3 个字母中任意一个的记录。SQL 代码如下。**（实例位置：光盘\TM\sl\9\9.36）**

```
SELECT * FROM info WHERE name REGEXP '[ceo]';
```

代码执行结果如图 9.52 所示。

```
mysql> select * from info where name regexp '[ceo]';
+----+--------+
| id | name   |
+----+--------+
|  1 | Aric   |
|  2 | Eric   |
|  4 | Jack   |
|  5 | Lucy   |
| 11 | abc12  |
|  6 | Bread  |
| 11 | abc12  |
+----+--------+
7 rows in set (0.01 sec)

mysql>
```

图 9.52　匹配指定字符中的任意一个

9.8.2　使用“*”和“+”来匹配多个字符

正则表达式中，“*”和“+”都可以匹配多个该符号之前的字符。但是，“+”至少表示一个字符，而“*”可以表示 0 个字符。

例 9.37　下面从 info 表 name 字段中查询字母‘c’之前出现过‘a’的记录。SQL 代码如下：**（实例位置：光盘\TM\sl\9\9.37）**

```
SELECT * FROM info WHERE name REGEXP 'a*c';
```

代码执行结果如图 9.53 所示。

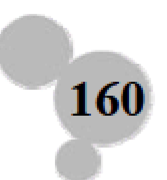
```
mysql> select * from info where name regexp 'a*c';
+----+--------+
| id | name   |
+----+--------+
|  1 | Aric   |
|  2 | Eric   |
|  4 | Jack   |
|  5 | Lucy   |
| 11 | abc12  |
+----+--------+
5 rows in set (0.00 sec)

mysql> _
```

图 9.53　使用“*”来匹配多个字符

查询结果显示，Aric、Eric 和 Lucy 中的字母 c 之前并没有 a。因为“*”可以表示 0 个，所以“a*c”表示字母 c 之前有 0 个或者多个 a 出现。上述的情况都是属于前面出现过 0 个的情况。如果使用“+”，其 SQL 代码如下：

```
SELECT * FROM info WHERE name REGEXP 'a+c';
```



代码执行结果如图 9.54 所示。

```
mysql> select * from info where name regexp 'a+c';
+----+------+
| id | name |
+----+------+
|  4 | Jack |
+----+------+
1 row in set (0.00 sec)

mysql> _
```

图 9.54　使用“+”来匹配多个字符

查询结果只有一条。只有 Jack 是刚好字母 c 前面出现了 a。因为 a+c 表示字母 c 前面至少有一个字母 a。

9.8.3　匹配以指定的字符开头和结束的记录

正则表达式中，^表示字符串的开始位置，$表示字符串的结束位置。下面将通过一个具体的实例演示如何匹配以指定的字符开头和结束的记录。

例 9.38　在学生成绩信息表 computer_stu 中查找姓名（name）字段中以 L 开头、以 y 结束的，中间包含两个字符的学生的成绩信息，运行结果如图 9.55 所示。**（实例位置：光盘\TM\sl\9\9.38）**

```
mysql> use db_database06;
Database changed
mysql> SELECT * FROM computer_stu WHERE name REGEXP '^L..y$';
+------+------+-------+
| id   | name | score |
+------+------+-------+
| 1001 | Lily |    85 |
| 1005 | Lucy |    65 |
+------+------+-------+
2 rows in set (0.01 sec)

mysql>
```

图 9.55　使用正则表达式查询学生成绩信息

要实现查询姓名（name）字段中以 L 开头、以 y 结束的，中间包含两个字符的学生成绩，可以通过正则表达式查询来实现。其中正则表达式中，^表示字符串的开始位置，$表示字符串的结束位置，. 表示除“\n”以外的任何单个字符，具体代码如下。

```
SELECT * FROM computer_stu WHERE name REGEXP '^L..y$';
```

9.9　小　　结

本章对 MySQL 数据库常见的查询方法进行了详细讲解，并通过大量的举例说明，帮助读者更好地理解所学知识的用法。在阅读本章时，读者应该重点掌握多条件查询、连接查询、子查询和查询结

果排序。本章学习的难点是使用正则表达式来查询。正则表达式的功能很强大，使用起来很灵活。希望读者能够阅读有关正则表达式的相关知识，能过对正则表达式了解得更加透彻。

9.10 实践与练习

1．实现从 computer_stu 表中查询获得一等奖学金的学生的学号、姓名和分数。各个等级的奖学金的最低分存储在 score 表中。**（答案位置：光盘\TM\sl\9\9.39）**

2．实现匹配 computer_stu 表中姓名字段中以 J 开头的记录。**（答案位置：光盘\TM\sl\9\9.40）**

第10章

常用函数

（视频讲解：36分钟）

MySQL数据库中提供了很丰富的函数。MySQL函数包括数学函数、字符串函数、日期和时间函数、条件判断函数、系统信息函数、加密函数、格式化函数等。函数的执行速度非常快，可以提高MySQL的处理速度，简化用户的操作。本章将详细介绍MySQL函数的相关知识。

通过阅读本章，读者可以：

- 了解MySQL函数
- 了解MySQL数学函数的使用方法
- 掌握MySQL字符串函数的使用方法
- 掌握MySQL日期和时间函数的使用方法
- 掌握MySQL条件判断函数的应用方法
- 掌握系统信息函数和加密函数的使用方法

10.1 MySQL 函数

MySQL 函数是 MySQL 数据库提供的内置函数。这些内置函数可以帮助用户更加方便地处理表中的数据。本节将简单地介绍 MySQL 中包含哪些类别的函数，以及这些函数的使用范围和作用，如表 10.1 所示。

表 10.1 MySQL 内置函数类别及作用

函　　数	作　　用
数学函数	用于处理数字。这类函数包括绝对值函数、正弦函数、余弦函数和获取随机数函数等
字符串函数	用于处理字符串。其中包括字符串连接函数、字符串比较函数、字符串中字母大小写转换函数等
日期和时间函数	用于处理日期和时间。其中包括获取当前时间的函数、获取当前日期的函数、返回年份的函数和返回日期的函数等
条件判断函数	用于在 SQL 语句中控制条件选择。其中包括 IF 语句、CASE 语句和 WHEN 语句等
系统信息函数	用于获取 MySQL 数据库的系统信息。其中包括获取数据库名的函数、获取当前用户的函数和获取数据库版本的函数等
加密函数	用于对字符串进行加密解密。其中包括字符串加密函数和字符串解密函数等
其他函数	包括格式化函数和锁函数等

MySQL 的内置函数不但可以在 SELECT 查询语句中应用，同样也可以在 INSERT、UPDATE 和 DELECT 等语句中应用。例如，在 INSERT 语句中，应用日期时间函数获取系统的当前时间，并且将其添加到数据表中。MySQL 内置函数可以对表中数据进行相应的处理，以便得到用户希望得到的数据。有了这些内置函数可以使 MySQL 数据库的功能更加强大。下面将对 MySQL 的内置函数逐一进行详细介绍。

10.2 数 学 函 数

数学函数是 MySQL 中常用的一类函数，主要用于处理数字，包括整型和浮点数等。MySQL 中内置的数学函数及其作用如表 10.2 所示。

表 10.2 MySQL 的数学函数

函　　数	作　　用
ABS(x)	返回 x 的绝对值
CEIL(x),CEILIN(x)	返回不小于 x 的最小整数值
FLOOR(x)	返回不大于 x 的最大整数值
RAND()	返回 0~1 的随机数
RAND(x)	返回 0~1 的随机数，x 值相同时返回的随机数相同

续表

函　数	作　用
SIGN(x)	返回参数作为-1、0 或 1 的符号，该符号取决于 x 的值为负、零或正
PI()	返回圆周率的值。默认的显示小数位数是 7 位，然而 MySQL 内部会使用完全双精度值
TRUNCATE(x,y)	返回数值 x 保留到小数点后 y 位的值
ROUND(x)	返回离 x 最近的整数
ROUND(x,y)	保留 x 小数点后 y 位的值，但截断时要进行四舍五入
POW(x,y),POWER(x,y)	返回 x 的 y 乘方的结果值
SQRT(x)	返回非负数 x 的二次方根
EXP(x)	返回 e 的 x 乘方后的值（自然对数的底）
MOD(x,y)	返回 x 除以 y 以后的余数
LOG(x)	返回 x 的基数为 2 的对数
LOG10(x)	返回 x 的基数为 10 的对数
RADIANS(x)	将角度转换为弧度
DEGREES(x)	返回参数 x，该参数由弧度转化为度
SIN(x)	返回 x 的正弦，其中 x 在弧度中被给定
ASIN(x)	返回 x 的反正弦，即正弦为 x 的值。若 x 不在-1~1 的范围之内，则返回 NULL
COS(x)	返回 x 的余弦，其中 x 在弧度上已知
ACOS(x)	返回 x 的反余弦，即余弦是 x 的值。若 x 不在-1~1 的范围之内，则返回 NULL
TAN(x)	返回 x 的反正切，即正切为 x 的值
ATAN(x),ATAN2(x,y)	返回两个变量 x 及 y 的反正切。它类似于 y 或 x 的反正切计算，除非两个参数的符号均用于确定结果所在象限
COT(x)	返回 x 的余切

下面对其中的常用函数进行讲解，并且配合以示例做详细说明。

10.2.1　ABS(x)函数

ABS(x)函数用于求绝对值。

例 10.1　使用 ABS(x)函数来求 5 和−5 的绝对值，其语句如下。**（实例位置：光盘\TM\sl\10\10.1）**

```
select ABS(5),ABS(-5);
```

查询结果如图 10.1 所示。

```
mysql> select ABS(5),ABS(-5);
+--------+---------+
| ABS(5) | ABS(-5) |
+--------+---------+
|      5 |       5 |
+--------+---------+
1 row in set (0.00 sec)

mysql>
```

图 10.1　使用 ABS(x)求数据的绝对值

10.2.2　FLOOR(x)函数

FLOOR(x)函数返回小于或等于 x 的最大整数。

例 10.2　应用 FLOOR(x)函数求小于或等于 1.5 及−2 的最大整数，其语句如下。（**实例位置：光盘\TM\sl\10\10.2**）

```
select FLOOR(1.5),FLOOR(-2);
```

其查询结果如图 10.2 所示。

```
mysql> select FLOOR(1.5),FLOOR(-2);
+------------+-----------+
| FLOOR(1.5) | FLOOR(-2) |
+------------+-----------+
|          1 |        -2 |
+------------+-----------+
1 row in set (0.00 sec)

mysql>
```

图 10.2　使用 FLOOR(x)函数求小于或等于数据的最大整数

10.2.3　RAND()函数

RAND()函数是返回 0~1 的随机数。

例 10.3　运用 RAND()函数，获取两个随机数，其语句如下。（**实例位置：光盘\TM\sl\10\10.3**）

```
Select RAND(),RAND();
```

其查询结果如图 10.3 所示。

```
mysql> select RAND(),RAND();
+------------------+------------------+
| RAND()           | RAND()           |
+------------------+------------------+
| 0.90826768419544 | 0.66054531993395 |
+------------------+------------------+
1 row in set (0.00 sec)

mysql>
```

图 10.3　使用 RAND()函数获取随机数



例 10.4　生成 3 个 1~100 之间的随机整数，效果如图 10.4 所示。（**实例位置：光盘\TM\sl\4\10.4**）

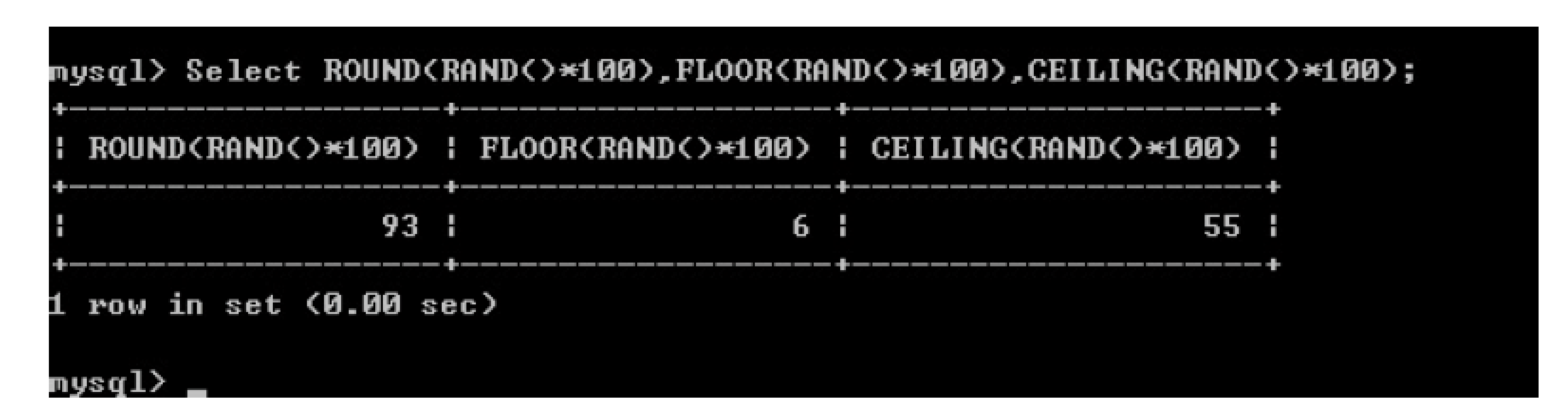


图 10.4　生成 3 个 1~100 之间的随机整数

使用 ROUND(x)生成一个与数 x 最接近的整数，当然，也可以使用 FLOOR(x)来生成一个小于或者等于 x 的最大整数。RAND()产生的是随机数，所以每次执行的结果都会是不一样的，具体代码如下。

```
Select ROUND(RAND()*100),FLOOR(RAND()*100),CEILING(RAND()*100);
```

10.2.4　PI()函数

PI()函数用于返回圆周率。

例 10.5　使用 PI()函数获取圆周率，其语句如下。（**实例位置：光盘\TM\sl\10\10.5**）

```
select PI();
```

查询结果如图 10.5 所示。

```
mysql> select PI();
+----------+
| PI()     |
+----------+
| 3.141593 |
+----------+
1 row in set (0.00 sec)

mysql>
```

图 10.5　使用 PI()函数获取圆周率

10.2.5　TRUNCATE(x,y)函数

TRUNCATE(x,y)函数返回 x 保留到小数点后 y 位的值。

例 10.6　使用 TRUNCATE(x,y)函数返回 2.123 456 7 小数点后 3 位的值，其语句如下。（**实例位置：光盘\TM\sl\10\10.6**）

```
select TRUNCATE(2.1234567,3);
```

其查询结果如图 10.6 所示。

```
mysql> select TRUNCATE(2.1234567,3);
+-----------------------+
| TRUNCATE(2.1234567,3) |
+-----------------------+
|                 2.123 |
+-----------------------+
1 row in set (0.02 sec)

mysql> _
```

图 10.6　使用 TRUNCATE(x,y)函数获取数据

10.2.6　ROUND(x)函数和 ROUND(x,y)函数

ROUND(x)函数返回离 x 最近的整数，也就是对 x 进行四舍五入处理；ROUND(x,y)函数返回 x 保留到小数点后 y 位的值，截断时需要进行四舍五入处理。

例 10.7　使用 ROUND(x)函数获取 1.6 和 1.2 最近的整数，使用 ROUND(x,y)函数获取 1.123 456 小数点后 3 位的值，其语句如下。**（实例位置：光盘\TM\sl\10\10.7）**

```
select ROUND(1.6),ROUND(1.2),ROUND(1.123456,3);
```

查询结果如图 10.7 所示。

```
mysql> select ROUND(1.6),ROUND(1.2),ROUND(1.123456,3);
+------------+------------+-------------------+
| ROUND(1.6) | ROUND(1.2) | ROUND(1.123456,3) |
+------------+------------+-------------------+
|          2 |          1 |             1.123 |
+------------+------------+-------------------+
1 row in set (0.02 sec)

mysql> _
```

图 10.7　使用 ROUND(x)函数和 ROUND(x,y)函数获取数据

10.2.7　SQRT(x)函数

SQRT(x)函数用于求平方根。

例 10.8　使用 SQRT(x)函数求 16 和 25 的平方根，其语句如下。**（实例位置：光盘\TM\sl\10\10.8）**

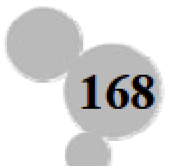
```
select SQRT(16),SQRT(25);
```

其查询结果如图 10.8 所示。

```
mysql> select SQRT(16),SQRT(25);
+----------+----------+
| SQRT(16) | SQRT(25) |
+----------+----------+
|        4 |        5 |
+----------+----------+
1 row in set (0.00 sec)

mysql> _
```

图 10.8　使用 SQRT(x)函数求 16 和 25 的平方根



10.3　字符串函数

字符串函数是 MySQL 中最常用的一类函数，主要用于处理表中的字符串，其作用如表 10.3 所示。

表 10.3　MySQL 的字符串函数

函　　数	作　　用
CHAR_LENGTH(s)	返回字符串 s 的字符数
LENGTH(s)	返回值为字符串 s 的长度，单位为字节。一个多字节字符算作多字节。这意味着对于一个包含 5 个 2 字节字符的字符串， LENGTH() 的返回值为 10，而 CHAR_LENGTH()的返回值则为 5
CONCAT(s1,s2, …)	返回结果为连接参数产生的字符串。如有任何一个参数为 NULL，则返回值为 NULL。或许有一个或多个参数。如果所有参数均为非二进制字符串，则结果为非二进制字符串。如果自变量中含有任一二进制字符串，则结果为一个二进制字符串。一个数字参数被转化为与之相等的二进制字符串格式；若要避免这种情况，可使用显式类型 cast，如 SELECT CONCAT(CAST(int_col AS CHAR), char_col)
CONCAT_WS(x,s1,s2, …)	同 CONCAT(s1,s2, …)函数，但是每个字符串直接要加上 x
INSERT(s1,x,len,s2)	将字符串 s2 替换 s1 的 x 位置开始长度为 len 的字符串
UPPER(s),UCASE(s)	将字符串 s 的所有字母都变成大写字母
LOWER(s),LCASE(s)	将字符串 s 的所有字母都变成小写字母
LEFT(s,n)	返回从字符串 s 开始的 n 最左字符
RIGHT(s,n)	从字符串 s 开始，返回最右 n 个字符
LPAD(s1,len,s2)	返回字符串 s1，其左边由字符串 s2 填补到 len 字符长度。假如 s1 的长度大于 len ，则返回值被缩短至 len 字符
RPAD(s1,len,s2)	返回字符串 s1，其右边被字符串 s2 填补至 len 字符长度。假如字符串 s1 的长度大于 len，则返回值被缩短到与 len 字符相同长度
LTRIM(s)	返回字符串 s，其引导空格字符被删除
RTRIM(s)	返回字符串 s，其结尾空格字符被删去
TRIM(s)	去掉字符串 s 开始处和结尾处的空格
TRIM(s1 FROM s)	去掉字符串 s 中开始处和结尾处的字符串 s1
REPEAT(s,n)	将字符串 s 重复 n 次
SPACE(n)	返回 n 个空格
REPLACE(s,s1,s2)	用字符串 s2 替代字符串 s 中的字符串 s1
STRCMP(s1,s2)	比较字符串 s1 和 s2

续表

函　　数	作　　用
SUBSTRING(s,n,len)	获取从字符串 s 中的第 n 个位置开始长度为 len 的字符串
MID(s,n,len)	同 SUBSTRING(s,n,len)
LOCATE(s1,s),POSITION(s1 IN s)	从字符串 s 中获取 s1 的开始位置
INSTR(s,s1)	查找字符串 s1 在 s 中的位置，返回首次出现位置的索引值
REVERSE(s)	将字符串 s 的顺序反过来
ELT(n,s1,s2,…)	返回第 n 个字符串
EXPORT_SET(bits,on,off[,separator [,number_of_bits]])	返回一个字符串，生成规则如下：针对 bits 的二进制格式，如果其位为 1，则返回一个 on 值；如果其位为 0，则返回一个 off 值。每个字符串使用 separator 进行分隔，默认值为“,”。number_of_bits 参数指定 bits 可用的位数，默认值为 64 位。例如，生成数字 182 的二进制（10110110）替换格式，以“@”作为分隔符，设置有效位为 6 位。其语句如下：select EXPORT_SET(182,'Y','N','@',6); 其运行结果为：N@Y@Y@N@Y@Y
FIELD(s,s1,s2, …)	返回第一个与字符串 s 匹配的字符串的位置
FIND_IN_SET(s1,s2)	返回在字符串 s2 中与 s1 匹配的字符串的位置
MAKE_SET(x,s1,s2, …)	按 x 的二进制数从 s1,s2, …,sn 中选取字符串

下面对其中的常用函数进行讲解，并且结合例子做详细说明。

10.3.1　INSERT(s1,x,len,s2)函数

INSERT(s1,x,len,s2)函数将字符串 s1 中 x 位置开始长度为 len 的字符串用字符串 s2 替换。

例 10.9　使用 INSERT 函数将 mrkj 字符串中的 kj 替换为 book，其语句如下。**（实例位置：光盘\TM\sl\10\10.9）**

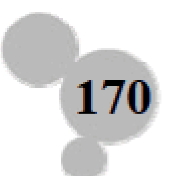
```
select INSERT('mrkej',3,2,'book');
```

替换后的查询结果如图 10.9 所示。

```
mysql> select INSERT('mrkj',3,2,'book');
+---------------------------+
| INSERT('mrkj',3,2,'book') |
+---------------------------+
| mrbook                    |
+---------------------------+
1 row in set (0.00 sec)

mysql>
```

图 10.9　使用 INSERT(s1,x,len,s2)函数替换指定字符串



10.3.2　UPPER(s)函数和 UCASE(s)函数

UPPER(s)函数和 UCASE(s)函数将字符串 s 的所有字母变成大写字母。

例 10.10　下面使用 UPPER(s)函数和 UCASE(s)函数将 mrbccd 字符串中的所有字母变成大写字母，其语句如下。**（实例位置：光盘\TM\sl\10\10.10）**

```
select UPPER('mrbccd'),UCASE('mrbccd');
```

其转换后的结果如图 10.10 所示。

```
mysql> select UPPER('mrbccd'),UCASE('mrbccd');
+-----------------+-----------------+
| UPPER('mrbccd') | UCASE('mrbccd') |
+-----------------+-----------------+
| MRBCCD          | MRBCCD          |
+-----------------+-----------------+
1 row in set (0.00 sec)

mysql> _
```

图 10.10　使用 UPPER(s)函数和 UCASE(s)函数将 mrbccd 字符串中的所有字母变成大写字母

10.3.3　LEFT(s,n)函数

LEFT(s,n)函数返回字符串 s 的前 n 个字符。

例 10.11　应用 LEFT 函数返回 mrbccd 字符串的前两个字符，其语句如下。**（实例位置：光盘\TM\sl\10\10.11）**

```
select LEFT('mrbccd',2);
```

其截取结果如图 10.11 所示。

```
mysql> select LEFT('mrbccd',2);
+------------------+
| LEFT('mrbccd',2) |
+------------------+
| mr               |
+------------------+
1 row in set (0.00 sec)

mysql>
```

图 10.11　使用 LEFT(s,n)函数返回指定字符

10.3.4　RTRIM(s)函数

RTRIM(s)函数将去掉字符串 s 结尾处的空格。

例 10.12 应用 RTRIM 函数去掉 mr 结尾处的空格，其语句如下。（**实例位置：光盘\TM\sl\10\10.12**）

```
select CONCAT('+',RTRIM(' mr '),'+');
```

其结果如图 10.12 所示。

```
mysql> select CONCAT('+',RTRIM(' mr '),'+');
+--------------------------------+
| CONCAT('+',RTRIM(' mr '),'+') |
+--------------------------------+
| + mr+                          |
+--------------------------------+
1 row in set (0.00 sec)

mysql>
```

图 10.12　使用 RTRIM(s)函数去掉 mr 结尾处的空格

10.3.5　SUBSTRING(s,n,len)函数

SUBSTRING(s,n,len)函数从字符串 s 的第 n 个位置开始获取长度为 len 的字符串。

例 10.13 下面使用 SUBSTRING 函数从 mrbccd 字符串的第三位开始获取 4 个字符，结果如图 10.13 所示。（**实例位置：光盘\TM\sl\10\10.13**）

```
mysql> select SUBSTRING('mrbccd',3,4);
+-------------------------+
| SUBSTRING('mrbccd',3,4) |
+-------------------------+
| bccd                    |
+-------------------------+
1 row in set (0.00 sec)

mysql>
```

图 10.13　使用 SUBSTRING(s,n,len)函数获取指定长度字符串

10.3.6　REVERSE(s)函数

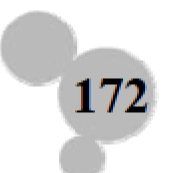
REVERSE(s)函数将字符串 s 的顺序反过来。

例 10.14 下面使用 REVERSE 函数将 mrbccd 字符串的顺序反过来，结果如图 10.14 所示。（**实例位置：光盘\TM\sl\10\10.14**）

```
mysql> select REVERSE('mrbccd');
+-------------------+
| REVERSE('mrbccd') |
+-------------------+
| dccbrm            |
+-------------------+
1 row in set (0.02 sec)

mysql>
```

图 10.14　使用 REVERSE(s)函数将 mrbccd 字符串的顺序反过来



10.3.7　FIELD(s,s1,s2,···)函数

FIELD(s,s1,s2,···)函数返回第一个与字符串 s 匹配的字符串的位置。

例 10.15　应用 FIELD 函数返回第一个与字符串 mr 匹配的字符串的位置，结果如图 10.15 所示。**（实例位置：光盘\TM\sl\10\10.15）**

```
mysql> select FIELD('mr','mrsoft','mrkj','mrbccd','mr');
+--------------------------------------------+
| FIELD('mr','mrsoft','mrkj','mrbccd','mr') |
+--------------------------------------------+
|                                          4 |
+--------------------------------------------+
1 row in set (0.00 sec)

mysql>
```

图 10.15　使用 FIELD(s,s1,s2,···)函数返回第一个与字符串 mr 匹配的字符串位置

10.3.8　LOCATE(s1,s)函数、POSITION(s1 IN s)函数和 INSTR(s,s1)函数

在 MySQL 中，可以通过 LOCATE(s1,s)、POSITION(s1 IN s)和 INSTR(s,s1)函数获取子字符串相匹配的开始位置。这 3 个函数的语法格式如下。

（1）LOCATE(s1,s)：表示子字符串 s1 和在字符串 s 中的开始位置。

（2）POSITION(s1 IN s)：表示子字符串 s1 在字符串 s 中的开始位置。

（3）INSTR(s,s1)：表示子字符串 s1 在字符串 s 中的开始位置。

注意

在使用这 3 个函数时，前两个函数 LOCATE(s1,s)和 POSITION(s1 IN s)的参数中，是把子字符串作为第一个参数，后第三个函数 INSTR(s,s1)则需要把子字符串作为第二个参数，这一点一定不要记错。

例 10.16　返回字符串“me”在字符串' You love me .He love me '中第一次出现的位置。效果如图 10.16 所示。**（实例位置：光盘\TM\sl\10\10.16）**

```
mysql> Select LOCATE('me','You love me.He love me.');
+-------------------------------------+
| LOCATE('me','You love me.He love me.') |
+-------------------------------------+
|                                  10 |
+-------------------------------------+
1 row in set (0.03 sec)

mysql> Select POSITION('me' IN 'You love me.He love me.');
+------------------------------------------+
| POSITION('me' IN 'You love me.He love me.') |
+------------------------------------------+
|                                       10 |
+------------------------------------------+
1 row in set (0.00 sec)

mysql> _
```

图 10.16　字符串函数的使用

对于字符串函数中的 LOCATE(s1,s),POSITION(s1 IN s)是从字符串 s 中获取 s1 的开始位置，具体代码如下。

```
Select LOCATE('me', 'You love me.He love me. ');
Select POSITION('me' IN 'You love me.He love me.');
```

10.4 日期和时间函数

日期和时间函数是 MySQL 中另一最常用的函数，主要用于对表中的日期和时间数据的处理。MySQL 内置的日期时间函数及作用如表 10.4 所示。

表 10.4 MySQL 的日期和时间函数

函 数	作 用
CURDATE(),CURRENT_DATE()	返回当前日期
CURTIME(),CURRENT_TIME()	返回当前时间
NOW(),CURRENT_TIMESTAMP(),LOCALTIME(),SYSDATE(),LOCALTIMESTAMP()	返回当前日期和时间
UNIX_TIMESTAMP()	以 UNIX 时间戳的形式返回当前时间
UNIX_TIMESTAMP(d)	将时间 d 以 UNIX 时间戳的形式返回
FROM_UNIXTIME(d)	把 UNIX 时间戳的时间转换为普通格式的时间
UTC_DATE()	返回 UTC（Universal Coordinated Time，国际协调时间）日期
UTC_TIME()	返回 UTC 时间
MONTH(d)	返回日期 d 中的月份值，范围是 1~12
MONTHNAME(d)	返回日期 d 中的月份名称，如 January、February 等
DAYNAME(d)	返回日期 d 是星期几，如 Monday、Tuesday 等
DAYOFWEEK(d)	返回日期 d 是星期几，1 表示星期日，2 表示星期一等
WEEKDAY(d)	返回日期 d 是星期几，0 表示星期一，1 表示星期二等
WEEK(d)	计算日期 d 是本年的第几个星期，范围是 0~53
WEEKOFYEAR(d)	计算日期 d 是本年的第几个星期，范围是 1~53
DAYOFYEAR(d)	计算日期 d 是本年的第几天
DAYOFMONTH(d)	计算日期 d 是本月的第几天
YEAR(d)	返回日期 d 中的年份值
QUARTER(d)	返回日期 d 是第几季度，范围是 1~4
HOUR(t)	返回时间 t 中的小时值
MINUTE(t)	返回时间 t 中的分钟值
SECOND(t)	返回时间 t 中的秒钟值

续表

函　　数	作　　用
EXTRACT(type FROM d)	从日期 d 中获取指定的值，type 指定返回的值，如 YEAR、HOUR 等将时间 t 转换为秒
TIME_TO_SEC(t)	将时间 t 转换为秒
SEC_TO_TIME(s)	将以秒为单位的时间 s 转换为时分秒的格式
TO_DAYS(d)	计算日期 d~0000 年 1 月 1 日的天数
FROM_DAYS(n)	计算从 0000 年 1 月 1 日开始 n 天后的日期
DATEDIFF(d1,d2)	计算日期 d1~d2 之间相隔的天数
ADDDATE(d,n)	计算起始日期 d 加上 n 天的日期
ADDDATE(d,INTERVAL expr type)	计算起始日期 d 加上一个时间段后的日期
DATE_ADD(d,INTERVAL expr type)	同 ADDDATE(d,INTERVAL n type)
SUBDATE(d,n)	计算起始日期 d 减去 n 天后的日期
SUBDATE(d,INTERVAL expr type)	计算起始日期 d 减去一个时间段后的日期
ADDTIME(t,n)	计算起始时间 t 加上 n 秒的时间
SUBTIME(t,n)	计算起始时间 t 减去 n 秒的时间
DATE_FROMAT(d,f)	按照表达式 f 的要求显示日期 d
TIME_FROMAT(t,f)	按照表达式 f 的要求显示时间 t
GET_FORMAT(type,s)	根据字符串 s 获取 type 类型数据的显示格式

10.4.1　CURDATE()函数和 CURRENT_DATE()函数

CURDATE()函数和 CURRENT_DATE()函数用于获取当前日期。

例 10.17　下面使用 CURDATE()函数和 CURRENT_DATE()函数获取当前日期，其语句如下。**（实例位置：光盘\TM\sl\10\10.17）**

```
select CURDATE(),CURRENT_DATE();
```

其查询结果如图 10.17 所示。

```
mysql> select CURDATE(),CURRENT_DATE();
+------------+----------------+
| CURDATE()  | CURRENT_DATE() |
+------------+----------------+
| 2011-07-08 | 2011-07-08     |
+------------+----------------+
1 row in set (0.03 sec)

mysql>
```

图 10.17　使用 CURDATE()和 CURRENT_DATE()函数获取当前日期

10.4.2　CURTIME()函数和 CURRENT_TIME()函数

CURTIME()函数和 CURRENT_TIME()函数用于获取当前时间。

例 10.18　下面使用 CURTIME()函数和 CURRENT_TIME()函数获取当前时间，其语句如下。**（实例位置：光盘\TM\sl\10\10.18）**

```
select CURTIME(),CURRENT_TIME();
```

其查询结果如图 10.18 所示。

```
mysql> select CURTIME(),CURRENT_TIME();
+-----------+----------------+
| CURTIME() | CURRENT_TIME() |
+-----------+----------------+
| 14:00:16  | 14:00:16       |
+-----------+----------------+
1 row in set (0.01 sec)

mysql> _
```

图 10.18　使用 CURTIME()函数和 CURRENT_TIME()函数获取当前时间

10.4.3　NOW()函数

NOW()函数获取当前日期和时间。还有 URRENT_TIMESTAMP()函数、LOCALTIME()函数、SYSDATE()函数和 LOCALTIMESTAMP()函数也同样可以获取当前日期和时间。

例 10.19　下面使用 NOW()函数、CURRENT_TIMESTAMP()函数、LOCALTIME()函数、SYSDATE()函数和 LOCALTIMESTAMP()函数来获取当前日期和时间，其语句如下。**（实例位置：光盘\TM\sl\10\10.19）**

```
select NOW(),CURRENT_TIMESTAMP(),LOCALTIME(),SYSDATE();
```

运行结果如图 10.19 所示。

```
--------+----------------------+
| NOW()                 | CURRENT_TIMESTAMP() | LOCALTIME()          | SYSDATE()
        | LOCALTIMESTAMP()     |
+---------------------+---------------------+---------------------+-------------
--------+----------------------+
| 2011-07-08 16:22:18 | 2011-07-08 16:22:18 | 2011-07-08 16:22:18 | 2011-07-08 1
6:22:18 | 2011-07-08 16:22:18 |
+---------------------+---------------------+---------------------+-------------
--------+----------------------+
1 row in set (0.00 sec)

mysql>
```

图 10.19　使用 NOW()、CURRENT_TIMESTAMP()等函数获取当前日期和时间



10.4.4　DATEDIFF(d1,d2)函数

DATEDIFF(d1,d2)用于计算日期 d1 与 d2 之间相隔的天数。

例 10.20　使用 DATEDIFF(d1,d2)函数计算 2011-07-05 与 2011-07-01 之间相隔的天数，其语句如下。**（实例位置：光盘\TM\sl\10\10.20）**

```
select DATEDIFF('2011-07-05','2011-07-01');
```

结果如图 10.20 所示。

```
mysql> select DATEDIFF('2011-07-05','2011-07-01');
+-------------------------------------+
| DATEDIFF('2011-07-05','2011-07-01') |
+-------------------------------------+
|                                   4 |
+-------------------------------------+
1 row in set (0.00 sec)

mysql> _
```

图 10.20　使用 DATEDIFF(d1,d2)函数计算 2011-07-05 与 2011-07-01 之间相隔的天数

10.4.5　ADDDATE(d,n)函数

ADDDATE(d,n)用于返回起始日期 d 加上 n 天的日期。

例 10.21　使用 ADDDATE(d,n)函数返回 2011-07-01 加上 3 天的日期，结果如图 10.21 所示。**（实例位置：光盘\TM\sl\10\10.21）**

```
mysql> select ADDDATE('2011-07-01',3);
+-------------------------+
| ADDDATE('2011-07-01',3) |
+-------------------------+
| 2011-07-04              |
+-------------------------+
1 row in set (0.00 sec)

mysql>
```

图 10.21　使用 ADDDATE(d,n)函数返回 2011-07-01 加上 3 天的日期

10.4.6　ADDDATE(d,INTERVAL expr type)函数

ADDDATE(d,INTERVAL expr type)函数返回起始日期 d 加上一个时间段后的日期。

例 10.22　使用 ADDDATE(d,INTERVAL expr type)函数返回 2011-07-01 加上一年两个月后的日期，其语句如下。**（实例位置：光盘\TM\sl\10\10.22）**

```
select ADDDATE('2011-07-01',INTERVAL,'12' YEAR_MONTH);
```

其运行结果如图 10.22 所示。

```
mysql> select ADDDATE('2011-07-01',INTERVAL '1 2' YEAR_MONTH);
+-------------------------------------------------+
| ADDDATE('2011-07-01',INTERVAL '1 2' YEAR_MONTH) |
+-------------------------------------------------+
| 2012-09-01                                      |
+-------------------------------------------------+
1 row in set (0.00 sec)

mysql>
```

图 10.22　使用 ADDDATE(d,INTERVAL expr type)函数返回 2011-07-01 加上一年两个月后的日期

10.4.7　SUBDATE(d,n)函数

SUBDATE(d,n)函数返回起始日期 d 减去 n 天的日期。

例 10.23　使用 SUBDATE(d,n)函数返回 2011-07-01 减去 6 天后的日期，结果如图 10.23 所示。（**实例位置：光盘\TM\sl\10\10.23**）

```
mysql> select SUBDATE('2011-07-01',6);
+-------------------------+
| SUBDATE('2011-07-01',6) |
+-------------------------+
| 2011-06-25              |
+-------------------------+
1 row in set (0.00 sec)

mysql>
```

图 10.23　使用 SUBDATE(d,n)函数返回 2011-07-01 减去 6 天后的日期

10.5　条件判断函数

条件函数用来在 SQL 语句中进行条件判断。根据不同的条件，执行不同的 SQL 语句。MySQL 支持的条件判断函数及作用如表 10.5 所示。

表 10.5　MySQL 的条件判断函数

函　　数	作　　用
IF(expr,v1,v2)	如果表达式 expr 成立，则执行 v1；否则执行 v2
IFNULL(v1,v2)	如果 v1 不为空，则显示 v1 的值；否则显示 v2 的值
CASE WHEN expr1 THEN v1 [WHEN expr2 THEN v2 …][ELSE vn] END	CASE 表示函数开始，END 表示函数结束。如果表达式 expr1 成立，则返回 v1 的值；如果表达式 expr2 成立，则返回 v2 的值。以此类推，最后遇到 else 时，返回 vn 的值。它的功能与 PHP 中的 switch 语句类似
CASE expr WHEN e1 THEN v1 [WHEN e2 THEN v2 …][ELSE vn] END	CASE 表示函数开始，END 表示函数结束。如果表达式 expr 取值为 e1，则返回 v1 的值；如果表达式 expr 取值为 e2，则返回 v2 的值，以此类推，最后遇到 ELSE，则返回 vn 的值

例 10.24　查询编程词典业绩信息表，如果业绩超过 100 万，则输出“Very Good”；如果业绩小于 100 万大于 10 万，则输出“Popularly”；否则输出“Not Good”。其语句如下。（**实例位置：光盘\TM\sl\10\10.24**）

```
select id,grade, CASE WHEN grade>1000000 THEN 'Very Good' WHEN grade<1000000 and grade >=100000
THEN 'Popularly' ELSE 'Not Good' END level from tb_bccd;
```

其查询结果如图 10.24 所示。

```
mysql> select id,grade, CASE WHEN grade>1000000 THEN 'Very Good' WHEN grade<1000
000 and grade >=100000 THEN 'Popularly' ELSE 'Not Good' END level from tb_bccd;
+----+---------+-----------+
| id | grade   | level     |
+----+---------+-----------+
|  1 | 2000000 | Very Good |
|  2 | 900000  | Popularly |
|  3 | 10000   | Not Good  |
+----+---------+-----------+
3 rows in set (0.00 sec)
```

图 10.24　条件判断函数的应用

10.6　系统信息函数

系统信息函数用来查询 MySQL 数据库的系统信息。例如，查询数据库的版本，查询数据库的当前用户等。如表 10.6 所示为各种系统信息函数的作用。

表 10.6　MySQL 的系统信息函数

函　数	作　用	示　例
VERSION()	获取数据库的版本号	select VERSION();
CONNECTION_ID()	获取服务器的连接数	select CONNECTION_ID();
DATABASE(),SCHEMA()	获取当前数据库名	select DATABASE(),SCHEMA();
USER(),SYSTEM_USER(),SESSION_USER()	获取当前用户	select USER(),SYSTEM_USER();
CURRENT_USER(),CURRENT_USER	获取当前用户	select CURRENT_USER();
CHARSET(str)	获取字符串 str 的字符集	select CHARSET('mrsoft');
COLLATION(str)	获取字符串 str 的字符排列方式	select COLLATION('mrsoft');
LAST_INSERT_ID()	获取最近生成的 AUTO_INCREMENT 值	select LAST_INSERT_ID();

10.6.1　获取 MySQL 版本号、连接数和数据库名的函数

VERSION()函数返回数据库的版本号；CONNECTION_ID()函数返回服务器的连接数，也就是到现在为止 MySQL 服务的连接次数；DATABASE()函数和 SCHEMA()函数返回当前数据库名。

例 10.25 下面将演示 VERSION()、CONNECTION_ID()、DATABASE()和 SCHEMA() 4 个函数的用法，如图 10.25 所示。（**实例位置：光盘\TM\sl\10\10.25**）

```
mysql> use test
Database changed
mysql> SELECT VERSION(),CONNECTION_ID();
+-----------+-----------------+
| VERSION() | CONNECTION_ID() |
+-----------+-----------------+
| 5.6.20    |               1 |
+-----------+-----------------+
1 row in set (0.00 sec)

mysql> SELECT DATABASE(),SCHEMA();
+------------+----------+
| DATABASE() | SCHEMA() |
+------------+----------+
| test       | test     |
+------------+----------+
1 row in set (0.00 sec)

mysql>
```

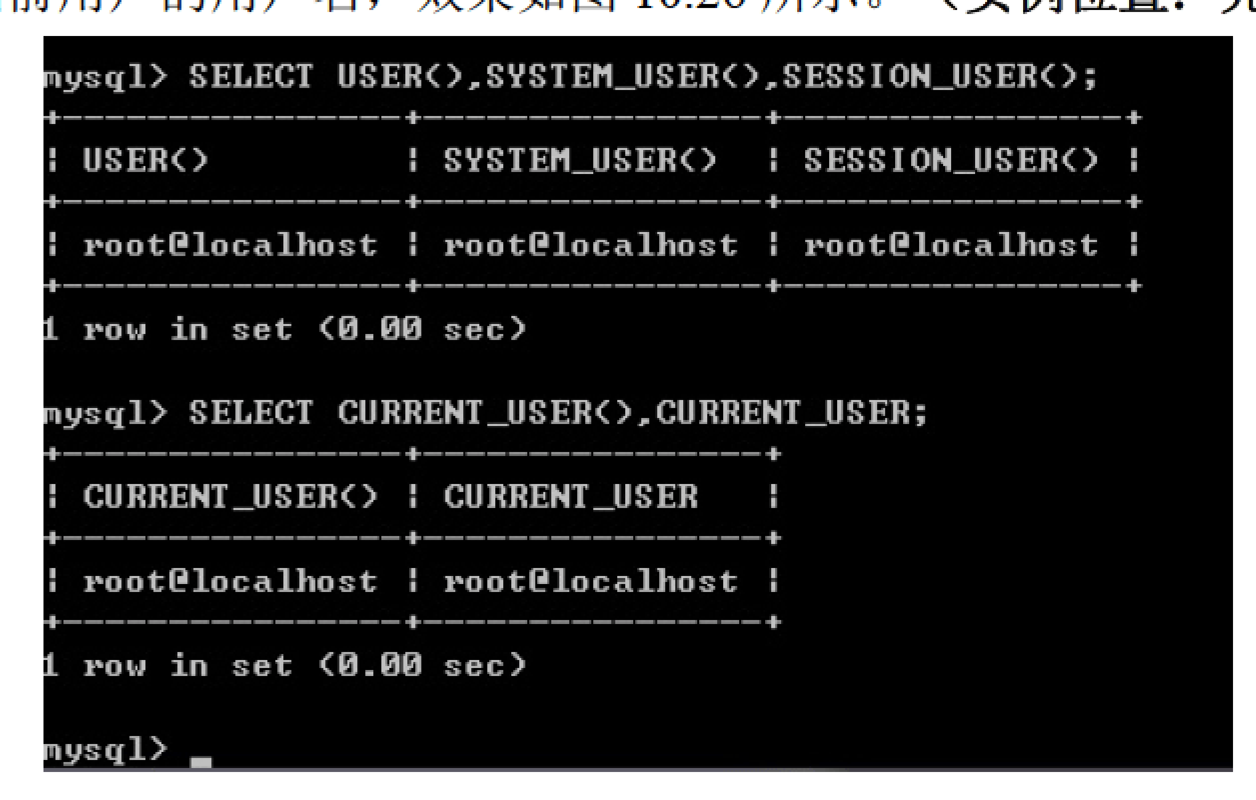
图 10.25　获取 MySQL 版本号、连接数和数据库名得函数

其中，VERSION()函数返回的版本号为“5.6.20”；CONNECTOIN_ID()函数返回的连接数为 1；DATABASE()函数和 SCHEMA()函数返回的当前数据库名是 test。

10.6.2　获取用户名的函数

USER()、SYSTEM_USER()、SESSION_USER()、CURRENT_USER()和 CURRENT_USER 这几个函数可以返回当前用户的名称。

例 10.26 查询当前用户的用户名，效果如图 10.26 所示。（**实例位置：光盘\TM\sl\10\10.26**）



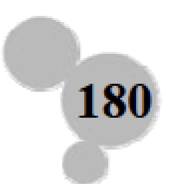
```
mysql> SELECT USER(),SYSTEM_USER(),SESSION_USER();
+----------------+----------------+----------------+
| USER()         | SYSTEM_USER()  | SESSION_USER() |
+----------------+----------------+----------------+
| root@localhost | root@localhost | root@localhost |
+----------------+----------------+----------------+
1 row in set (0.00 sec)

mysql> SELECT CURRENT_USER(),CURRENT_USER;
+----------------+----------------+
| CURRENT_USER() | CURRENT_USER   |
+----------------+----------------+
| root@localhost | root@localhost |
+----------------+----------------+
1 row in set (0.00 sec)

mysql>
```

图 10.26　获取用户名的函数

结果显示，当前用户的用户名为 root。localhost 是主机名。因为服务器和客户端在一台机器上，所以服务器的主机名为 localhost。用户名和主机名之间用符号“@”进行连接。



10.6.3　获取字符串的字符集和排序方式的函数

CHARSET(str)函数返回字符串 str 的字符集，一般情况下这个字符集就是系统的默认字符集；COLLATION(str)函数返回字符串 str 的字符排列方式。

例 10.27　查看字符串'aa'的字符集和字符串排序方式，效果如图 10.27 所示。（**实例位置：光盘\TM\sl\10\10.27**）

```
mysql> SELECT CHARSET('aa'),COLLATION('aa');
+---------------+-----------------+
| CHARSET('aa') | COLLATION('aa') |
+---------------+-----------------+
| utf8          | utf8_general_ci |
+---------------+-----------------+
1 row in set (0.01 sec)

mysql> _
```

图 10.27　获取字符串的字符集和排序方式的函数

10.7　加密函数

加密函数是 MySQL 中用来对数据进行加密的函数。因为数据库中有些很敏感的信息不希望被其他人看到，所以就可以通过加密的方式来使这些数据变成看似乱码的数据。例如，用户密码就应该进行加密。各种加密函数的作用如表 10.7 所示。

表 10.7　MySQL 的加密函数

函　　数	作　　用	示　　例
PASSWORD(str)	对字符串 str 进行加密。经此函数加密后的数据是不可逆的。其经常用于对用户注册的密码进行加密处理	对字符串 mrsoft 进行加密，其语句如下： select PASSWORD('mrsoft');
MD5(str)	对字符串 str 进行加密。经常用于对普通数据进行加密	使用 MD5()函数对 mrsoft 字符串进行加密，其语句如下： select MD5('mrsoft');
ENCODE(str,pswd_str)	使用字符串 pswd_str 来加密字符串 str。加密的结果是一个二进制数，必须使用 BLOB 类型的字段来保存它	使用字符串 mr 对 mrsoft 进行加密处理，其语句如下： select ENCODE('mrsoft','mr');
DECODE(crypt_str,pswd_str)	使用字符串 pswd_str 来为 crypt_str 解密。crypt_str 是通过 ENCODE(str,pswd_str)加密后的二进制数据。字符串 pswd_str 应该与加密时的字符串 pswd_str 是相同的	应用 DECODE 函数对经过 ENCODE 函数加密的字符串进行解密，其语句如下： select DECODE(ENCODE('mrsoft','mr'),'mr');

10.7.1　加密函数 PASSWORD(str)

PASSWORD(str)函数可以对字符串 str 进行加密。一般情况下，PASSWORD(str)函数主要是用来给用户的密码加密的。

例 10.28　使用 PASSWORD(str)函数为字符串'abcd'加密，效果如图 10.28 所示。（**实例位置：光盘\TM\sl\10\10.28**）



图 10.28　加密函数 PASSWORD(str)

结果显示，字符串“abcd”加密后的结果是“*A154C2565E9E7F94BFC08A1FE702624ED8EFDA”。PASSWORD(str)函数加密是不可逆的。


技巧

PASSWORD(str)函数经常用来给密码加密。MySQL 用户需要设置密码，用户不能将未加密的密码直接存储到 MySQL 的 user 表中。因为登录 MySQL 数据库时，数据库系统会将输入的密码先通过 PASSWORD(str)函数加密，然后与数据库中的密码进行比较，匹配成功后才可以登录。

10.7.2　加密函数 MD5(str)

MD5(str)函数可以对字符串 str 进行加密。MD5(str)函数主要对普通的数据进行加密。

例 10.29　使用 MD5(str)函数为字符串'abcd'加密，效果如图 10.29 所示。（**实例位置：光盘\TM\sl\10\10.29**）

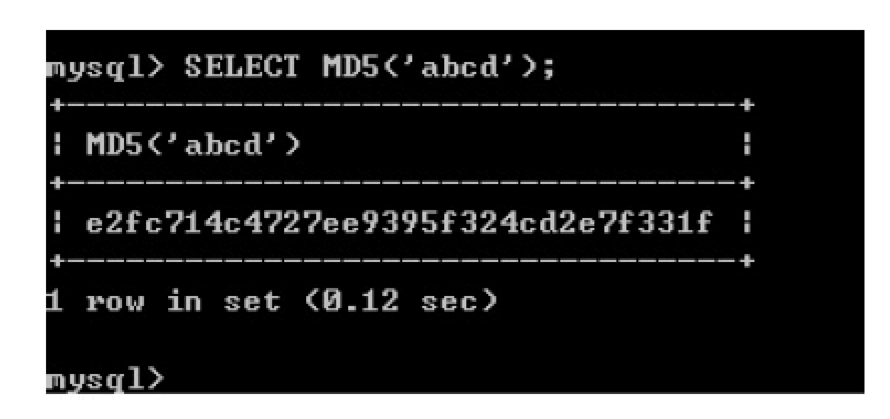
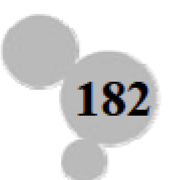


图 10.29　加密函数 MD5(str)

结果显示，字符串 abcd 的 MD5 值为 e2fc714c4727ee9395f324cd2e7f331f。



10.8　其 他 函 数

MySQL 中除了上述内置函数以外，还包含很多函数。例如，数字格式化函数 FORMAT(x,n)，IP 地址与数字的转换函数 INET_ATON(ip)，还有加锁函数 GET_LOCT(name,time)、解锁函数 RELEASE_LOCK(name)等。在表 10.8 中罗列了 MySQL 中支持的其他函数。

表 10.8　MySQL 的其他函数

函　　数	作　　用
FORMAT(x,n)	将数字 x 进行格式化，将 x 保留到小数点后 n 位。这个过程需要进行四舍五入
ASCII(s)	ASCII(s)返回字符串 s 的第一个字符的 ASCII 码
BIN(x)	BIN(x)返回 x 的二进制编码
HEX(x)	HEX(x)返回 x 的十六进制编码
OCT(x)	OCT(x)返回 x 的八进制编码
CONV(x,f1,f2)	CONV(x,f1,f2)将 x 从 f1 进制数变成 f2 进制数
INET_ATON(IP)	INET_ATON(IP)函数可以将 IP 地址转换为数字表示
INET_NTOA(N)	INET_NTOA(N)函数可以将数字 n 转换成 IP 的形式
GET_LOCT(name,time)	GET_LOCT(name,time)函数定义一个名称为 name、持续时间长度为 time 秒的锁。锁定成功，返回 1；如果尝试超时，返回 0；如果遇到错误，返回 NULL
RELEASE_LOCK(name)	RELEASE_LOCK(name)函数解除名称为 name 的锁。如果解锁成功，返回 1；如果尝试超时，返回 0；如果解锁失败，返回 NULL
IS_FREE_LOCK(name)	IS_FREE_LOCK(name)函数判断是否使用名为 name 的锁。如果使用，返回 0；否则返回 1
BENCHMARK(count,expr)	将表达式 expr 重复执行 count 次，然后返回执行时间。该函数可以用来判断 MySQL 处理表达式的速度
CONVERT(s USING cs)	将字符串 s 的字符集变成 cs
CAST(x AS type)	将 x 变成 type 类型，这两个函数只对 BINARY、CHAR、DATE、DATETIME、TIME、SIGNED INTEGER、UNSIGNED INTEGER 这些类型起作用。但两种方法只是改变了输出值的数据类型，并没有改变表中字段的类型

10.8.1　格式化函数 FORMAT(x,n)

FORMAT(x,n)函数可以将数字 x 进行格式化，将 x 保留到小数点后 n 位。这个过程需要进行四舍五入。例如，FORMAT(2.356,2)返回的结果将会是 2.36；FORMAT(2.353,2)返回的结果将会是 2.35。

例 10.30　使用 FORMAT(x,n)函数来将 235.345 6 和 235.345 4 进行格式化，都保留到小数点后 3 位，效果如图 10.30 所示。**（实例位置：光盘\TM\sl\10\10.30）**

```
mysql> SELECT FORMAT(235.3456,3),FORMAT(234.3454,3);
+--------------------+--------------------+
| FORMAT(235.3456,3) | FORMAT(234.3454,3) |
+--------------------+--------------------+
| 235.346            | 234.345            |
+--------------------+--------------------+
1 row in set (0.03 sec)

mysql>
```

图 10.30 格式化函数 FORMAT

结果显示，235.345 6 格式化后的结果是 235.346；235.345 4 格式化后的结果是 235.345。这个数都保留到小数点后 3 位，而且都进行了四舍五入处理。

注意

FORMAT(x,n)函数可以将 x 保留到小数点后 n 位。在格式化过程中需要进行四舍五入的操作。FORMAT(x,n)函数与 ROUND(x,y)函数返回 x 保留到小数点后 y 位的值。截断时需要进行四舍五入处理。

10.8.2 改变字符集的函数

CONVERT(s USING cs)函数将字符串 s 的字符集变成 cs。

例 10.31 将字符串'ABC'的字符集变成 gbk，效果如图 10.31 所示。**（实例位置：光盘\TM\sl\10\10.31）**

```
mysql> SELECT CHARSET('ABC'),CHARSET(CONVERT('ABC' USING gbk));
+----------------+-----------------------------------+
| CHARSET('ABC') | CHARSET(CONVERT('ABC' USING gbk)) |
+----------------+-----------------------------------+
| utf8           | gbk                               |
+----------------+-----------------------------------+
1 row in set (0.02 sec)

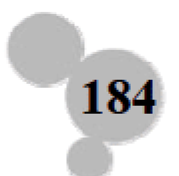
mysql>
```

图 10.31 改变字符集的函数

10.8.3 改变字段数据类型的函数

CAST(x AS type)和 CONVERT(x,type)这两个函数将 x 变成 type 类型。这两个函数只对 BINARY、CHAR、DATETIME、TIME、SIGNED INTEGER、UNSIGNED INTEGER 这些类型起作用。但两种方法只是改变了输出值的数据类型，并没有改变表中字段的类型。

例 10.32 下面 C1 表中的 times 字段为 DATETIME 类型，将其变为 DATE 类型或者 TIME 类型，效果如图 10.32 所示。**（实例位置：光盘\TM\sl\10\10.32）**



```
mysql> USE test;
Database changed
mysql> SELECT times,CAST(times as DATE),CONVERT(times,TIME) FROM C1;
+---------------------+---------------------+---------------------+
| times               | CAST(times as DATE) | CONVERT(times,TIME) |
+---------------------+---------------------+---------------------+
| 2014-09-16 11:05:30 | 2014-09-16          | 11:05:30            |
+---------------------+---------------------+---------------------+
1 row in set (0.01 sec)

mysql>
```

图 10.32 改变字段数据类型的函数

结果显示，times 字段原来的取值是 2014-09-16 11:05:30，这是 DATETIME 类型；CAST(times AS DATE)返回的结果是 2014-09-16，这说明类型已经变成了 DATE 型；CONVERT(times,TIME)返回的结果是 11:05:30，这说明类型已经变成了 TIME 类型。

10.9 小 结

本章介绍了 MySQL 数据库提供的内部函数，包括数学函数、字符串函数、日期和时间函数、条件判断函数、系统信息函数和加密函数等。字符串函数、日期和时间函数是本章的重点内容；条件判断函数是本章的难点，因为条件判断函数涉及很多条件判断和跳转的语句。这些函数通常与 SELECT 语句一起使用，用来方便用户的查询。同时，INSERT、UPDATE、DELECT 语句和条件表达式也可以使用这些函数。

10.10 实践与练习

1．编写 SQL 语句，实现利用函数来查看当前数据库的版本号、当前数据库的名称和当前的用户。**（答案位置：光盘\TM\sl\10\10.33）**

2．编写 SQL 语句，应用 INSTR()函数返回字符串“me”在字符串“You love me .He love me”中第一次出现的位置。**（答案位置：光盘\TM\sl\10\10.34）**

第 11 章

索　引

（视频讲解：22 分钟）

索引是一种特殊的数据库结构，是提高数据库性能的重要方式，可以用来快速查询数据库表中的特定记录，MySQL 中所有的数据类型都可以被索引。MySQL 的索引包括普通索引、唯一性索引、全文索引、单列索引、多列索引和空间索引等。本章将介绍索引的概念、作用、不同类别，用不同的方法创建索引以及删除索引的方法等。

通过阅读本章，读者可以：

- 了解 MySQL 索引的概念
- 了解 MySQL 索引的分类
- 掌握在建立数据表时创建索引的方法
- 掌握在已建立的数据表中创建索引的方法
- 掌握修改数据表结构添加索引的方法
- 掌握删除索引的方法

11.1　索 引 概 述

在 MySQL 中，索引由数据表中一列或多列组合而成，创建索引的目的是为了优化数据库的查询速度。其中，用户创建的索引指向数据库中具体数据所在位置。当用户通过索引查询数据库中的数据时，不需要遍历所有数据库中的所有数据，大幅度提高了查询效率。

11.1.1　MySQL 索引概述

索引是一种将数据库中单列或者多列的值进行排序的结构。应用索引，可以大幅度提高查询的速度。

用户通过索引查询数据，不但可以提高查询速度，也可以降低服务器的负载。用户查询数据时，系统可以不必遍历数据表中的所有记录，而是查询索引列。一般过程的数据查询是通过遍历全部数据，并寻找数据库中的匹配记录而实现的。与一般形式的查询相比，索引就像一本书的目录。而当用户通过目录查找书中内容时，就好比用户通过目录查询某章节的某个知识点。这样就为用户在查找内容过程中，缩短大量时间，帮助用户有效地提高查找速度。所以，使用索引可以有效地提高数据库系统的整体性能。

应用 MySQL 数据库时，并非用户在查询数据的时候，总需要应用索引来优化查询。凡事都有双面性，使用索引可以提高检索数据的速度，对于依赖关系的子表和父表之间的联合查询时，可以提高查询速度，并且可以提高整体的系统性能。但是，创建索引和维护需要耗费时间，并且该耗费时间与数据量的大小成正比；另外，索引需要占用物理空间，给数据的维护造成很多麻烦。

整体来说，索引可以提高查询的速度，但是会影响用户操作数据库的插入操作。因为，向有索引的表中插入记录时，数据库系统会按照索引进行排序。所以，用户可以将索引删除后，插入数据，当数据插入操作完成后，用户可以重新创建索引。

说明

不同的存储引擎定义每个表的最大索引数和最大索引长度。所有存储引擎对每个表至少支持 16 个索引。总索引长度至少为 256 字节。有些存储引擎支持更多的索引数和更大的索引长度。索引有两种存储类型，包括 B 树（BTREE）索引和哈希（HASH）索引。其中，B 树为系统默认索引方法。

11.1.2　MySQL 索引分类

MySQL 的索引包括普通索引、唯一性索引、全文索引、单列索引、多列索引和空间索引等。

1. 普通索引

普通索引，即不应用任何限制条件的索引，该索引可以在任何数据类型中创建。字段本身的约束条件可以判断其值是否为空或唯一。创建该类型索引后，用户在查询时，便可以通过索引进行查询。在某数据表的某一字段中，建立普通索引后。用户需要查询数据时，只需根据该索引进行查询即可。

2. 唯一性索引

使用 UNIQUE 参数可以设置唯一索引。创建该索引时，索引的值必须唯一，通过唯一索引，用户可以快速定位某条记录，主键是一种特殊唯一索引。

3. 全文索引

使用 FULLTEXT 参数可以设置索引为全文索引。全文索引只能创建在 CHAR、VARCHAR 或者 TEXT 类型的字段上。查询数据量较大的字符串类型的字段时，使用全文索引可以提高查询速度。例如，查询带有文章回复内容的字段，可以应用全文索引方式。需要注意的是，在默认情况下，应用全文搜索大小写不敏感。如果索引的列使用二进制排序后，可以执行大小写敏感的全文索引。

4. 单列索引

顾名思义，单列索引即只对应一个字段的索引。其可以包括上述叙述的 3 种索引方式。应用该索引的条件只需要保证该索引值对应一个字段即可。

5. 多列索引

多列索引是在表的多个字段上创建一个索引。该索引指向创建时对应的多个字段，用户可以通过这几个字段进行查询。要想应用该索引，用户必须使用这些字段中的第一个字段。

6. 空间索引

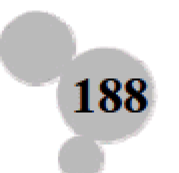
使用 SPATIAL 参数可以设置索引为空间索引。空间索引只能建立在空间数据类型上，这样可以提高系统获取空间数据的效率。MySQL 中只有 MyISAM 存储引擎支持空间检索，而且索引的字段不能为空值。

11.2 创建索引

创建索引是指在某个表中至少一列中建立索引，以便提高数据库性能。其中，建立索引可以提高表的访问速度。本节通过几种不同的方式创建索引。其中包括在建立数据库时创建索引、在已经建立的数据表中创建索引和修改数据表结构创建索引。

11.2.1 在建立数据表时创建索引

在建立数据表时可以直接创建索引，这种方式比较直接，且方便、易用。在建立数据表时创建索



引的基本语法结构如下。

```
create table table_name(
属性名  数据类型[约束条件],
属性名  数据类型[约束条件]
...
属性名  数据类型
[UNIQUE | FULLTEXT | SPATIAL ]   INDEX }KEY
[别名](  属性名 1 [(长度)] [ASC | DESC])
);
```

其中，属性名后的属性值，其含义如下。

（1）UNIQUE：可选项，表明索引为唯一性索引。

（2）FULLTEXT：可选项，表明索引为全文搜索。

（3）SPATIAL：可选项，表明索引为空间索引。

INDEX 和 KEY 参数用于指定字段索引，用户在选择时，只需要选择其中的一种即可；另外别名为可选项，其作用是给创建的索引取新名称；别名的参数如下。

（1）属性名 1：指索引对应的字段名称，该字段必须被预先定义。

（2）长度：可选项，指索引的长度，必须是字符串类型才可以使用。

（3）ASC/DESC：可选项，ASC 表示升序排列，DESC 参数表示降序排列。

1．普通索引创建

创建普通索引，即不添加 UNIQUE、FULLTEXT 等任何参数。

例 11.1　下面创建表名为 score 的数据表，并在该表的 id 字段上建立索引，其主要代码如下。**（实例位置：光盘\TM\sl\11\11.1）**

```
create table score(
id int(11) auto_increment primary key not null,
name varchar(50) not null,
math int(5) not null,
english int(5) not null,
chinese int(5) not null,
index(id));
```

运行以上代码的结果如图 11.1 所示。

```
mysql> create table score(
    -> id int(11) auto_increment primary key not null,
    -> name varchar(50) not null,
    -> math int(5) not null,
    -> english int(5) not null,
    -> chinese int(5) not null,
    -> index(id));
Query OK, 0 rows affected (0.08 sec)
```

图 11.1　创建普通索引

在命令提示符中使用 SHOW CREATE TABLE 语句查看该表的结构，在命令提示符中输入的代码

如下。

```
show create table score;
```

其运行结果如图 11.2 所示。

```
| score | CREATE TABLE `score` (
  `id` int(11) NOT NULL auto_increment,
  `name` varchar(50) NOT NULL,
  `math` int(5) NOT NULL,
  `english` int(5) NOT NULL,
  `chinese` int(5) NOT NULL,
  PRIMARY KEY (`id`),
  KEY `id` (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 |
```

索 引 值

图 11.2　查看数据表结构

从图 11.2 中可以清晰地看到，该表结构的索引为 id，则可以说明该表的索引建立成功。

2．创建唯一性索引

创建唯一性索引与创建一般索引的语法结构大体相同，但是在创建唯一索引的时候，需要使用 UNIQUE 参数进行约束。

例 11.2　创建一个表名为 address 的数据表，并指定该表的 id 字段上建立唯一索引，其代码如下所示。**（实例位置：光盘\TM\sl\11\11.2）**

```
create table address(
id int(11) auto_increment primary key not null,
name varchar(50),
address varchar(200),
UNIQUE INDEX address(id ASC));
```

应用 SHOW CREATE TABLE 语句查看表的结构，其运行如图 11.3 所示。

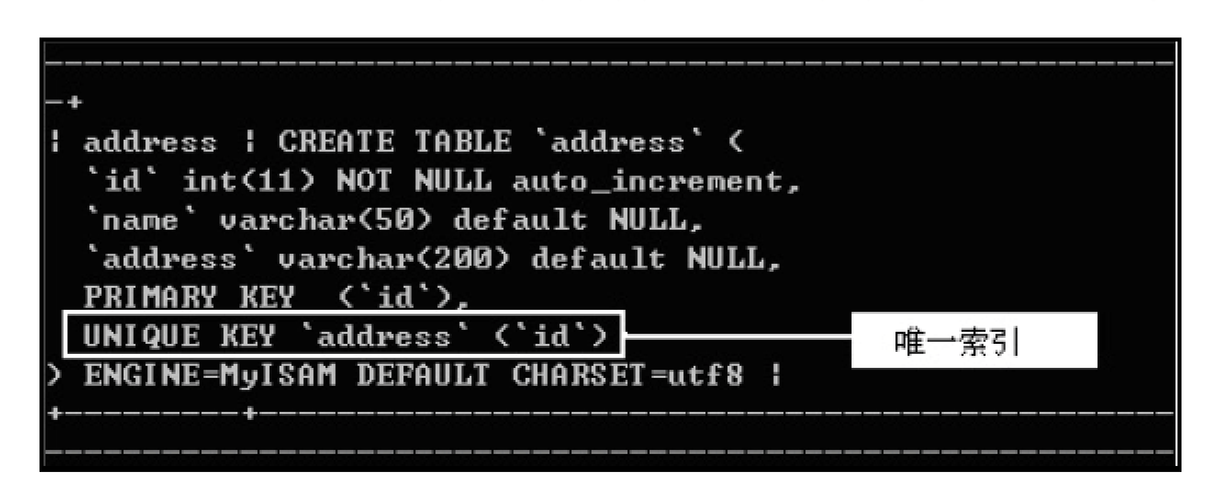


图 11.3　查看唯一索引的表结构

从图 11.3 中可以看到，该表的 id 字段上已经建立了一个名为 address 的唯一索引。

说明

虽然添加唯一索引可以约束字段的唯一性，但是有时候并不能提高用户查找速度，即不能实现优化查询目的。所以，读者在使用过程中需要根据实际情况来选择唯一索引。



3. 创建全文索引

与创建普通索引和唯一索引不同，全文索引的创建只能作用在 CHAR、VARCHAR、TEXT 类型的字段上。创建全文索引需要使用 FULLTEXT 参数进行约束。

例 11.3　创建一个名称为 cards 的数据表，并在该表的 number 字段上创建全文索引，其代码如下。**（实例位置：光盘\TM\sl\11\11.3）**

```
create table cards(
id int(11) auto_increment primary key not null,
name varchar(50),
number bigint(11),
info varchar(50),
FULLTEXT KEY cards_info(info)) engine=MyISAM;
```

在命令提示符中应用 SHOW CREATE TABLE 语句查看表结构，其代码如下。

```
SHOW CREATE TABLE cards;
```

运行结果如图 11.4 所示。

```
+-------+------------------------------------
-------------------------------------------
| cards | CREATE TABLE `cards` (
  `id` int(11) NOT NULL auto_increment,
  `name` varchar(50) default NULL,
  `number` bigint(11) default NULL,
  `info` varchar(50) default NULL,
  PRIMARY KEY  (`id`),
  FULLTEXT KEY `cards_info` (`info`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 |
+-------+------------------------------------
-------------------------------------------
```

图 11.4　查看全文索引的数据表结构



只有 MyISAM 类型的数据表支持 FULLTEXT 全文索引，InnoDB 或其他类型的数据表不支持全文索引。当用户在建立全文索引的时候，返回 “ERROR 1283 (HY000): Column 'number' cannot be part of FULLTEXT index” 的错误，则说明用户操作的当前数据表不支持全文索引，即不为 MyISAM 类型的数据表。

4. 创建单列索引

创建单列索引，即在数据表的单个字段上创建索引。创建该类型索引不需要引入约束参数，用户在建立时只需指定单列字段名，即可创建单列索引。

例 11.4　创建名称为 telephone 的数据表，并指定在 tel 字段上建立名称为 tel_num 的单列索引，其代码如下。**（实例位置：光盘\TM\sl\11\11.4）**

```
create table telephone(
id int(11) primary key auto_increment not null,
name varchar(50) not null,
```

```
tel varchar(50) not null,
index tel_num(tel(20))
);
```

运行上述代码后，应用 SHOW CREATE TABLE 语句查看表的结构，其运行结果如图 11.5 所示。

```
| telephone | CREATE TABLE `telephone` (
  `id` int(11) NOT NULL auto_increment,
  `name` varchar(50) NOT NULL,
  `tel` varchar(50) NOT NULL,
  PRIMARY KEY  (`id`),
  KEY `tel_num` (`tel`(20))
) ENGINE=MyISAM DEFAULT CHARSET=utf8 |
```

图 11.5　查看单列索引表的数据表结构

说明

数据表中的字段长度为 50，而创建的索引的字段长度为 20，这样做的目的是为了提高查询效率，优化查询速度。

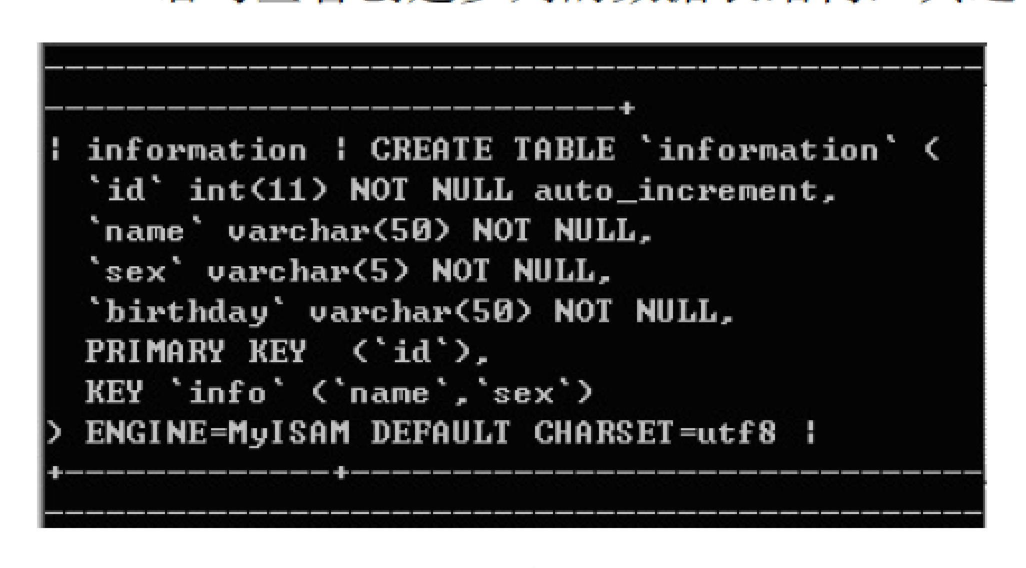
5. 创建多列索引

与创建单列索引相仿，创建多列索引即指定表的多个字段即可实现。

例 11.5　创建名称为 information 的数据表，并指定 name 和 sex 为多列索引，其代码如下。**（实例位置：光盘\TM\sl\11\11.5）**

```
create table information(
id int(11) auto_increment primary key not null,
name varchar(50) not null,
sex varchar(5) not null,
birthday varchar(50) not null,
INDEX info(name,sex)
);
```

应用 SHOW CREATE TABLE 语句查看创建多列的数据表结构，其运行结果如图 11.6 所示。



```
| information | CREATE TABLE `information` (
  `id` int(11) NOT NULL auto_increment,
  `name` varchar(50) NOT NULL,
  `sex` varchar(5) NOT NULL,
  `birthday` varchar(50) NOT NULL,
  PRIMARY KEY  (`id`),
  KEY `info` (`name`,`sex`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 |
```

图 11.6　查看多列索引表的数据结构

需要注意的是，在多列索引中，只有查询条件中使用了这些字段中的第一个字段（即上面示例中的 name 字段）时，索引才会被使用。





说明

触发多列索引的条件是用户必须使用索引的第一字段，如果没有用到第一字段，则索引不起任何作用，用户想要优化查询速度，可以应用该类索引形式。

6. 创建空间索引

创建空间索引时，需要设置 SPATIAL 参数。同样，必须说明的是，只有 MyISAM 类型表支持该类型索引。而且，索引字段必须有非空约束。

例 11.6　创建一个名称为 list 的数据表，并创建一个名为 listinfo 的空间索引，其代码如下。**（实例位置：光盘\TM\sl\11\11.6）**

```
create table list(
id int(11) primary key auto_increment not null,
goods geometry not null,
SPATIAL INDEX listinfo(goods)
)engine=MyISAM;
```

运行上述代码，创建成功后，在命令提示符中应用 SHOW CREATE TABLE 语句查看表的结构。其运行结果如图 11.7 所示。

```
+--------+------------------------------------------
| list   | CREATE TABLE `list` (
  `id` int(11) NOT NULL auto_increment,
  `goods` GEOMETRY    NOT NULL,
  PRIMARY KEY  (`id`)
  SPATIAL KEY `listinfo`(`goods`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 |
+--------+------------------------------------------
```

图 11.7　查看空间索引表的结构

从图 11.7 中可以看到，goods 字段上已经建立名称为 listinfo 的空间索引，其中，goods 字段必须不能为空，且数据类型是 GEOMETRY。该类型是空间数据类型。空间类型不能用其他类型代替，否则在生成空间索引时会产生错误且不能正常创建该类型索引。

说明

空间类型除了上述示例中提到的 GEOMETRY 类型外，还包括如 POINT、LINESTRING、POLYGON 等类型。这些空间数据类型在平常的操作中很少被用到。

11.2.2　在已建立的数据表中创建索引

在 MySQL 中，不但可以在用户创建数据表时创建索引，用户也可以直接在已经创建的表中，在已经存在的一个或几个字段上创建索引。其基本的命令结构如下所示。

```
CREATE [UNIQUE | FULLTEXT |SPATIAL ] INDEX index_name
ON table_name(属性 [(length)] [ ASC | DESC]);
```

命令的参数说明如下。

（1）index_name 为索引名称，该参数作用是给用户创建的索引赋予新的名称。

（2）table_name 为表名，即指定创建索引的表名称。

（3）可选参数，指定索引类型，包括 UNIQUE（唯一索引）、FULLTEXT（全文索引）、SPATIAL（空间索引）。

（4）属性参数，指定索引对应的字段名称。该字段必须已经预存在用户想要操作的数据表中，如果该数据表中不存在用户指定的字段，则系统会提示异常。

（5）length 为可选参数，用于指定索引长度。

（6）ASC 和 DESC 参数，指定数据表的排序顺序。

与建立数据表时创建索引相同，在已建立的数据表中创建索引同样包含 6 种索引方式。

1. 创建普通索引

例 11.7 首先，应用 SHOW CREATE TABLE 语句查看 studentinfo 表的结构，其运行结果如图 11.8 所示。**（实例位置：光盘\TM\sl\11\11.7）**

```
| studentinfo | CREATE TABLE `studentinfo` (
  `sid` int(11) NOT NULL auto_increment,
  `name` varchar(50) NOT NULL,
  `age` varchar(11) NOT NULL,
  `sex` varchar(2) NOT NULL default 'M',
  `tel` bigint(11) NOT NULL,
  `time` varchar(50) NOT NULL,
  PRIMARY KEY  (`sid`),
  KEY `index_name` (`name`),
  KEY `index_student_info` (`name`,`sex`)
) ENGINE=MyISAM AUTO_INCREMENT=10 DEFAULT CHARSET=utf8 |
```

图 11.8　查看未添加索引前的表结构

然后，在该表中创建名称为 stu_info 的普通索引，在命令提示符中输入如下命令。

```
create INDEX stu_info ON studentinfo(sid);
```

输入上述命令后，应用 SHOW CREATE TABLE 语句查看该数据表的结构。其运行结果如图 11.9 所示。

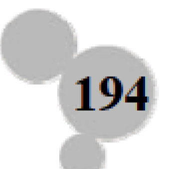
```
| studentinfo | CREATE TABLE `studentinfo` (
  `sid` int(11) NOT NULL auto_increment,
  `name` varchar(50) NOT NULL,
  `age` varchar(11) NOT NULL,
  `sex` varchar(2) NOT NULL default 'M',
  `tel` bigint(11) NOT NULL,
  `time` varchar(50) NOT NULL,
  PRIMARY KEY  (`sid`),
  KEY `index_name` (`name`),
  KEY `index_student_info` (`name`,`sex`),
  KEY `stu_info` (`sid`)                          stu_info索引
) ENGINE=MyISAM AUTO_INCREMENT=10 DEFAULT CHARSET=utf8 |
```

图 11.9　查看添加索引后的表格结构

从图 11.9 中可以看出，名称为 stu_info 的数据表创建成功。如果系统没有提示异常或错误，则说明已经向 studentinfo 数据表中建立名称为 stu_info 的普通索引。

2. 创建唯一索引

在已经存在的数据表中建立唯一索引的命令如下。



```
CREATE UNIQUE INDEX 索引名 ON 数据表名称(字段名称);
```

其中，UNIQUE 是用来设置索引唯一性的参数，该表中的字段名称既可以存在唯一性约束，也可以不存在唯一性约束。

例 11.8　下面在 index1 表中的 cid 字段上建立名为 index1_id 的唯一性索引，代码如下。（**实例位置：光盘\TM\sl\11\11.8**）

```
CREATE UNIQUE INDEX index1_id ON index1(cid);
```

输入上述命令后，应用 SHOW CREATE TABLE 语句查看该数据表的结构。其运行结果如图 11.10 所示。

```
------------------------------------------+
| index1 | CREATE TABLE `index1` (
  `cid` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(50) COLLATE utf8_unicode_ci NOT NULL,
  `age` varchar(11) COLLATE utf8_unicode_ci NOT NULL,
  `sex` varchar(2) COLLATE utf8_unicode_ci NOT NULL,
  `tel` bigint(11) NOT NULL,
  `time` varchar(50) COLLATE utf8_unicode_ci NOT NULL,
  PRIMARY KEY (`cid`),
  UNIQUE KEY `index1_id` (`cid`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci |
+--------+----------------------------------------------------
------------------------------------------+
1 row in set (0.00 sec)

mysql>
```

图 11.10　查看添加唯一索引后的表格结构

3．创建全文索引

在 MySQL 中，为已经存在的数据表创建全文索引的命令如下。

```
CREATE FULLTEXT INDEX 索引名 ON 数据表名称(字段名称);
```

其中，FULLTEXT 用来设置索引为全文索引。操作的数据表类型必须为 MyISAM 类型。字段类型必须为 VARCHAR、CHAR、TEXT 等类型。

例 11.9　下面在 index2 表中的 info 字段上建立名为 index2_info 的全文索引，代码如下。（**实例位置：光盘\TM\sl\11\11.9**）

```
CREATE FULLTEXT INDEX index2_info ON index2(info);
```

输入上述命令后，应用 SHOW CREATE TABLE 语句查看该数据表的结构。其运行结果如图 11.11 所示。

```
--------------+
| index2 | CREATE TABLE `index2` (
  `cid` int(11) NOT NULL AUTO_INCREMENT,
  `info` varchar(50) COLLATE utf8_unicode_ci NOT NULL,
  PRIMARY KEY (`cid`),
  FULLTEXT KEY `index2_info` (`info`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci |
+--------+-----------------------------------------------------
--------------+
1 row in set (0.00 sec)

mysql>
```

图 11.11　查看添加全文索引后的表格结构

4．创建单列索引

与建立数据表时创建单列索引相同，用户可以设置单列索引。其命令结构如下。

```
CREATE INDEX 索引名 ON 数据表名称(字段名称(长度));
```

设置字段名称长度，可以优化查询速度，提高查询效率。

例 11.10　下面在 index3 表中的 address 字段上建立名为 index3_addr 的单列索引。Address 字段的数据类型为 varchar(20)，索引的数据类型为 char(4)，代码如下。**（实例位置：光盘\TM\sl\11\11.10）**

```
CREATE INDEX index3_addr ON index3(address(4));
```

输入上述命令后，应用 SHOW CREATE TABLE 语句查看该数据表的结构。其运行结果如图 11.12 所示。

```
-------------+
| index3 | CREATE TABLE `index3` (
  `cid` int(11) NOT NULL AUTO_INCREMENT,
  `address` varchar(20) COLLATE utf8_unicode_ci NOT NULL,
  PRIMARY KEY (`cid`),
  KEY `index3_addr` (`address`(4))
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci |
+--------+------------------------------------------------------------
--------------------------------------------------------------------
```

图 11.12　查看添加单列索引后的表格结构

5．创建多列索引

建立多列索引与建立单列索引类似。其主要命令结构如下。

```
CREATE INDEX 索引名 ON 数据表名称(字段名称 1,字段名称 2,…);
```

与建立数据表时创建多列索引相同，当创建多列索引时，用户必须使用第一字段作为查询条件，否则索引不能生效。

例 11.11　下面在 index4 表中的 name 和 address 字段上建立名为 index4_na 的多列索引，代码如下。**（实例位置：光盘\TM\sl\11\11.11）**

```
CREATE INDEX index4_na ON index4(name,address);
```

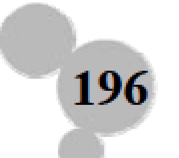
输入上述命令后，应用 SHOW CREATE TABLE 语句查看该数据表的结构。其运行结果如图 11.13 所示。

```
--------------------------------------------------------------------
| index4 | CREATE TABLE `index4` (
  `cid` int(11) NOT NULL AUTO_INCREMENT,
  `name` varchar(20) COLLATE utf8_unicode_ci NOT NULL,
  `address` varchar(20) COLLATE utf8_unicode_ci NOT NULL,
  PRIMARY KEY (`cid`),
  KEY `index4_na` (`name`,`address`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci |
+--------+----------------------------------------------------------
--------------------------------------------------------------------
```

图 11.13　查看添加多列索引后的表格结构

6．创建空间索引

建立空间索引，用户需要应用 SPATIAL 参数作为约束条件。其命令结构如下。



```
CREATE SPATIAL   INDEX 索引名 ON 数据表名称(字段名称);
```

其中，SPATIAL 用来设置索引为空间索引。用户要操作的数据表类型必须为 MyISAM 类型。并且字段名称必须存在非空约束，否则将不能正常创建空间索引。

11.2.3　修改数据表结构添加索引

修改已经存在表上的索引，可以通过 ALTER TABLE 语句为数据表添加索引，其基本结构如下。

```
ALTER TABLE table_name ADD [ UNIQUE | FULLTEXT |SPATIAL ] INDEX index_name(属性名 [(length)]
[ASC | DESC]);
```

该参数与 11.2.1 节和 11.2.2 节中所介绍的参数相同，这里不再赘述，请读者参阅前面两节中的内容。

1. 添加普通索引

首先，应用 SHOW CREATE TABLE 语句查看 studentinfo 表的结构，其运行结果如图 11.14 所示。

```
| studentinfo | CREATE TABLE `studentinfo` (
  `sid` int(11) NOT NULL auto_increment,
  `name` varchar(50) NOT NULL,
  `age` varchar(11) NOT NULL,
  `sex` varchar(2) NOT NULL default 'M',
  `tel` bigint(11) NOT NULL,
  `time` varchar(50) NOT NULL,
  PRIMARY KEY  (`sid`),
  KEY `index_name` (`name`),
  KEY `index_student_info` (`name`,`sex`),
  KEY `stu_info` (`sid`)
) ENGINE=MyISAM AUTO_INCREMENT=10 DEFAULT CHARSET=utf8 |
```

图 11.14　查看未添加索引前的表结构

然后，在该表中添加名称为 timer 的普通索引，在命令提示符中输入如下命令。

```
alter table studentinfo ADD INDEX timer (time(20));
```

输入上述命令后，应用 SHOW CREATE TABLE 语句查看该数据表的结构。其运行结果如图 11.15 所示。

```
| studentinfo | CREATE TABLE `studentinfo` (
  `sid` int(11) NOT NULL auto_increment,
  `name` varchar(50) NOT NULL,
  `age` varchar(11) NOT NULL,
  `sex` varchar(2) NOT NULL default 'M',
  `tel` bigint(11) NOT NULL,
  `time` varchar(50) NOT NULL,
  PRIMARY KEY  (`sid`),
  KEY `index_name` (`name`),
  KEY `index_student_info` (`name`,`sex`),
  KEY `stu_info` (`sid`),
  KEY `timer` (`time`(20))          名称为timer索引
) ENGINE=MyISAM AUTO_INCREMENT=10 DEFAULT CHARSET=utf8 |
```

图 11.15　查看添加索引后的表格结构

从图 11.15 中可以看出，名称为 timer 的数据表添加成功，已经成功向 studentinfo 数据表中添加名称为 timer 的普通索引。


说明

从功能上看，修改数据表结构添加索引与在已存在数据表中建立索引所实现功能大体相同，二者均是在已经建立的数据表中添加或创建新的索引。所以，用户在使用的时候，可以根据个人需求和实际情况，选择适合的方式向数据表中添加索引。

2. 添加唯一索引

与已存在的数据表中添加索引的过程类似，在数据表中添加唯一索引的命令结构如下所示。

```
ALTER TABLE 表名 ADD UNIQUE INDEX 索引名称*(字段名称);
```

其中， ALTER 语句一般是用来修改数据表结构的语句，ADD 为添加索引的关键字；UNIQUE 是用来设置索引唯一性的参数，该表中的字段名称既可以存在唯一性约束，也可以不存在唯一性约束。

3. 添加全文索引

创建全文索引与创建普通索引和唯一索引不同，全文索引创建只能作用在 CHAR、VARCHAR、TEXT 类型的字段上。创建全文索引需要使用 FULLTEXT 参数进行约束。

在 MySQL 中，为已经存在的数据表添加全文索引的命令如下。

```
ALTER TABLE 表名 ADD  FULLTEXT INDEX 索引名称(字段名称);
```

其中，ADD 是添加的关键字，FULLTEXT 用来设置索引为全文索引。操作的数据表类型必须为 MyISAM 类型。字段类型同样必须为 VARCHAR、CHAR、TEXT 等类型。

例 11.12 使用 ALTER INDEX 语句在数据表 workinfo 的 address 字段上创建名为 index_ext 的全文索引。**（实例位置：光盘\TM\sl\11\11.12）**

用修改数据表结果的方式添加全文索引。用 ALTER INDEX 语句在 address 字段上创建名为 index_ext 的全文索引，具体代码如下。

```
ALTER TABLE workinfo ADD FULLTEXT INDEX index_ext(address);
```

输入上述命令后，应用 SHOW CREATE TABLE 语句查看该数据表的结构。其运行结果如图 11.16 所示。

```
| workinfo | CREATE TABLE `workinfo` (
  `id` int(10) NOT NULL AUTO_INCREMENT,
  `name` varchar(20) COLLATE utf8_unicode_ci NOT NULL,
  `address` varchar(50) COLLATE utf8_unicode_ci DEFAULT NULL,
  `tel` varchar(20) COLLATE utf8_unicode_ci DEFAULT NULL,
  PRIMARY KEY (`id`),
  UNIQUE KEY `index_id` (`id`),
  KEY `index_name` (`name`(10)),
  FULLTEXT KEY `index_ext` (`address`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 COLLATE=utf8_unicode_ci |
+----------+---------------------------------------------------
```

图 11.16 查看使用 ALTER TABLE 语句创建的全文索引

4. 添加单列索引

与建立数据表时创建单列索引相同，用户可以设置单列索引。其命令结构如下。



```
ALTER TABLE 表名 ADD  INDEX 索引名称(字段名称(长度));
```

同样，用户可以设置字段名称长度，以便优化查询速度，提高执行效率。

5．添加多列索引

添加多列索引与建立单列索引类似。其主要命令结构如下。

```
ALTER TABLE 表名 ADD  INDEX 索引名称(字段名称 1,字段名称 2,…);
```

使用 ALTER 修改数据表结构同样可以添加多列索引。与建立数据表时创建多列索引相同，当创建多列索引时，用户必须使用第一字段作为查询条件，否则索引不能生效。

6．添加空间索引

添加空间索引，用户需要应用 SPATIAL 参数作为约束条件。其命令结构如下。

```
ALTER TABLE 表名 ADD  SPATIAL INDEX 索引名称(字段名称);
```

其中，SPATIAL 用来设置索引为空间索引。用户要操作的数据表类型必须为 MyISAM 类型，并且字段名称必须存在非空约束，否则将不能正常创建空间索引。该类别索引并不常用，初学者只需要了解该索引类型即可。

11.3　删 除 索 引

在 MySQL 中，创建索引后，如果用户不再需要该索引，则可以删除指定表的索引。因为这些已经被建立且不常使用的索引，一方面可能会占用系统资源，另一方面也可能导致更新速度下降，这极大地影响了数据表的性能。所以，在用户不需要该表的索引时，可以手动删除指定索引。其中，删除索引可以通过 DROP 语句来实现。其基本的命令如下。

```
DROP INDEX index_name ON table_name;
```

其中，参数 index_name 是用户需要删除的索引名称，参数 table_name 指定数据表名称，下面应用示例向读者展示如何删除数据表中已经存在的索引。打开 MySQL 后，应用 SHOW CREATE TABLE 语句查看数据表的索引，其运行结果如图 11.17 所示。

```
--------------------------------------------

--------------------------------------------
-+
| address | CREATE TABLE `address` (
  `id` int(11) NOT NULL auto_increment,
  `name` varchar(50) default NULL,
  `address` varchar(200) default NULL,
  PRIMARY KEY  (`id`),
  UNIQUE KEY `address` (`id`)
) ENGINE=MyISAM DEFAULT CHARSET=utf8 |
+----------+--------------------------------

--------------------------------------------
```

图 11.17　查看 address 数据表内的索引

从图 11.17 中可以看出，名称为 address 的数据表中存在唯一索引 address。在命令提示符中继续输入如下命令。

```
DROP INDEX id ON address
```

运行上述代码的结果如图 11.18 所示。

在用户顺利删除索引后，为确定该索引是否已被删除，用户可以再次应用 SHOW CREATE TABLE 语句来查看数据表结构。其运行结果如图 11.19 所示。

```
mysql> drop index address on address;
Query OK, 0 rows affected (0.13 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

图 11.18　删除唯一索引 address

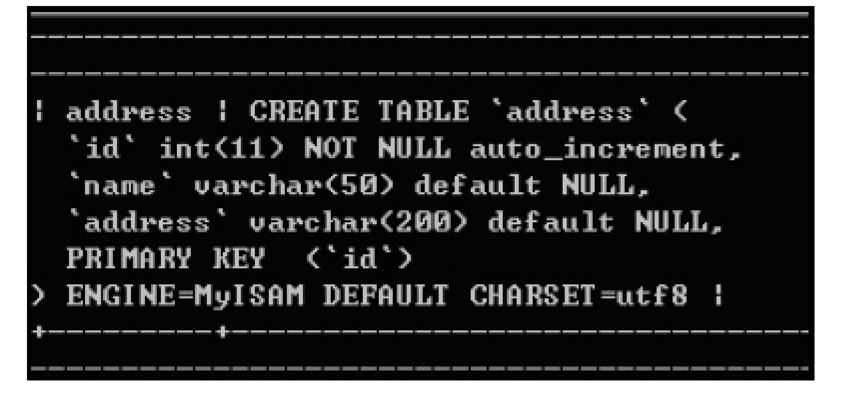


图 11.19　再次查看 address 数据表结构

从图 11.19 可以看出，名称为 address 的唯一索引已经被删除。

例 11.13　本实例将使用 DROP 语句从数据表中删除不再需要的索引，效果如图 11.20 所示。**（实例位置：光盘\TM\sl\11\11.13）**

```
mysql> DROP INDEX index_id ON workinfo;
Query OK, 0 rows affected (0.03 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql>
```

图 11.20　删除唯一性索引

使用 DROP 语句删除 workinfo 表的唯一性索引 index_id，具体代码如下。

```
DROP INDEX index_id ON workinfo;
```

11.4　小　　结

本章对 MySQL 数据库的索引的基础知识、创建索引、删除索引进行了详细讲解，创建索引的内容是本章的重点。读者应该重点掌握创建索引的 3 种方法，分别为创建表的时候创建索引、使用 CREATE INDEX 语句来创建索引和使用 ALTER TABLE 语句来创建索引。

11.5　实践与练习

1．运用 CREATE INDEX 语句为 name 字段创建长度为 10 的索引 index_name。**（答案位置：光盘\TM\sl\11\11.14）**

2．创建一个表名为 tb_user 的数据表，并在该表的 id 字段上建立唯一索引。**（答案位置：光盘\TM\sl\11\11.15）**



第12章

视　　图

（视频讲解：22 分钟）

视图是从一个或多个表中导出的表，是一种虚拟存在的表。视图就像一个窗口，通过这个窗口可以看到系统专门提供的数据。这样，用户可以不用看到整个数据库表中的数据，而只关心对自己有用的数据。视图可以使用户的操作更方便，而且可以保障数据库系统的安全性。本章将介绍视图的含义和作用，视图定义的原则和创建视图的方法，并对修改视图、查看视图和删除视图的方法进行了详细的讲解。

通过阅读本章，读者可以：

- 了解使用 CREATE VIEW 语句创建视图的方法
- 了解创建视图的注意事项
- 掌握使用 SHOW TABLE STATUS 语句查看视图的方法
- 掌握使用 CREATE OR REPLACE VIEW 语句修改视图的方法
- 掌握使用 ALTER 语句修改视图的方法
- 掌握更新视图和使用 DROP VIEW 语句删除视图的方法

12.1 视图概述

视图是由数据库中的一个表或多个表导出的虚拟表，方便用户对数据的操作。本节将详细讲解视图的概念及作用。

12.1.1 视图的概念

视图是一个虚拟表，是从数据库中一个或多个表中导出来的表，其内容由查询定义。同真实的表一样，视图包含一系列带有名称的列和行数据。但是，数据库中只存放了视图的定义，而并没有存放视图中的数据。这些数据存放在原来的表中。使用视图查询数据时，数据库系统会从原来的表中取出对应的数据。因此，视图中的数据是依赖于原来的表中的数据的。一旦表中的数据发生改变，显示在视图中的数据也会发生改变。

视图是存储在数据库中的查询的 SQL 语句，它主要出于两种原因：安全原因，视图可以隐藏一些数据，例如，员工信息表，可以用视图只显示姓名、工龄、地址，而不显示社会保险号和工资数等；另一个原因是可使复杂的查询易于理解和使用。

12.1.2 视图的作用

对其中所引用的基础表来说，视图的作用类似于筛选。定义视图的筛选可以来自当前或其他数据库的一个或多个表，或者其他视图。通过视图进行查询没有任何限制，通过它们进行数据修改时的限制也很少。下面将视图的作用归纳为如下几点。

1. 简单性

看到的就是需要的。视图不仅可以简化用户对数据的理解，也可以简化他们的操作。那些被经常使用的查询可以被定义为视图，从而使得用户不必为以后的操作每次指定全部的条件。

2. 安全性

视图的安全性可以防止未授权用户查看特定的行或列，使有权限用户只能看到表中特定行的方法如下。

（1）在表中增加一个标志用户名的列。

（2）建立视图，使用户只能看到标有自己用户名的行。

（3）把视图授权给其他用户。

3. 逻辑数据独立性

视图可以使应用程序和数据库表在一定程度上独立。如果没有视图，程序一定是建立在表上的。有了视图之后，程序可以建立在视图之上，从而程序与数据库表被视图分割开来。视图可以在以下几



个方面使程序与数据独立。

（1）如果应用建立在数据库表上，当数据库表发生变化时，可以在表上建立视图，通过视图屏蔽表的变化，从而使应用程序可以不动。

（2）如果应用建立在数据库表上，当应用发生变化时，可以在表上建立视图，通过视图屏蔽应用的变化，从而使数据库表不动。

（3）如果应用建立在视图上，当数据库表发生变化时，可以在表上修改视图，通过视图屏蔽表的变化，从而使应用程序可以不动。

（4）如果应用建立在视图上，当应用发生变化时，可以在表上修改视图，通过视图屏蔽应用的变化，从而使数据库可以不动。

12.2　创 建 视 图

创建视图是指在已经存在的数据库表上建立视图，视图可以建立在一张表中，也可以建立在多张表中。本节主要讲解创建视图的方法。

12.2.1　查看创建视图的权限

创建视图需要具有 CREATE VIEW 的权限，同时应该具有查询涉及的列的 SELECT 权限。可以使用 SELECT 语句来查询这些权限信息，查询语法如下。

```
SELECT Selete_priv,Create_view_priv FROM mysql.user WHERE user='用户名';
```

（1）Selete_priv 属性表示用户是否具有 SELECT 权限，Y 表示拥有 SELECT 权限，N 表示没有。

（2）Create_view_priv 属性表示用户是否具有 CREATE VIEW 权限；mysql.user 表示 MySQL 数据库下面的 user 表。

（3）“用户名”参数表示要查询是否拥有 DROP 权限的用户，该参数需要用单引号引起来。

例 12.1　下面查询 MySQL 中 root 用户是否具有创建视图的权限，代码如下。**（实例位置：光盘\TM\sl\12\12.1）**

```
SELECT Select_priv,Create_view_priv FROM mysql.user WHERE user='root';
```

执行结果如图 12.1 所示。



图 12.1　查看用户是否具有创建视图的权限

结果中 Select_priv 和 Create_view_priv 属性的值都为 Y，这表示 root 用户具有 SELECT 和 CREATE VIEW 权限。

12.2.2 创建视图的步骤

MySQL 中，创建视图是通过 CREATE VIEW 语句实现的，其语法如下。

```
CREATE [ALGORITHM={UNDEFINED|MERGE|TEMPTABLE}]
        VIEW 视图名[(属性清单)]
        AS SELECT 语句
        [WITH [CASCADED|LOCAL] CHECK OPTION];
```

（1）ALGORITHM 是可选参数，表示视图选择的算法；

（2）“视图名”参数表示要创建的视图名称；

（3）“属性清单”是可选参数，指定视图中各个属性的名词，默认情况下与 SELECT 语句中查询的属性相同；

（4）SELECT 语句参数是一个完整的查询语句，表示从某个表中查出某些满足条件的记录，将这些记录导入视图中；

（5）WITH CHECK OPTION 是可选参数，表示更新视图时要保证在该视图的权限范围之内。

例 12.2 在数据表 tb_book 中创建 view1 视图，视图命名为 book_view1，并设置视图属性分别为 a_sort、a_talk、a_books，代码如下。**（实例位置：光盘\TM\sl\12\12.2）**

```
CREATE VIEW
book_view1(a_sort,a_talk,a_books)
AS SELECT sort,talk,books
FROM tb_book;
```

执行结果如图 12.2 所示。

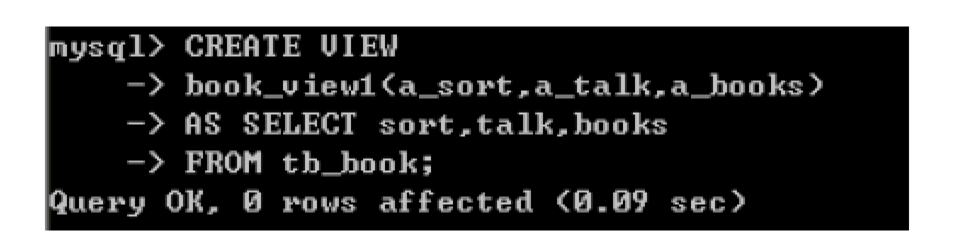


图 12.2 创建视图 book_view1

如果要在 tb_book 表和 tb_user 表上创建名为 book_view1 的视图，执行代码如下。

```
CREATE ALGORITHM=MERGE VIEW
book_view1(a_sort,a_talk,a_books,a_name)
AS SELECT sort,talk,books,tb_user.name
FROM tb_book,tb_name WHERE tb_book.id=tb_name.id
WITH LOCAL CHECK OPTION;
```

例 12.3 在实际项目开发过程中的数据表中可能有很多的字段，但某个模块可能只需要其中的几个。为了提高查询速度和简化操作，可以将该模块需要的字段单独提取出来放在某个视图中，例如，本实例涉及学生表和成绩表，在建立的视图中只含有与学生成绩有关的字段，如图 12.3 所示。**（实例位置：光盘\TM\sl\12\12.3）**



```
mysql> create view scoreinfo as select sno,sname,yw,wy,sx from tb_student,tb_sco
re where tb_student.id=tb_score.sid;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from scoreinfo;
+---------+--------+----+----+----+
| sno     | sname  | yw | wy | sx |
+---------+--------+----+----+----+
| 0312315 | 刘小华 | 88 | 60 | 94 |
| 0312316 | 金星星 | 60 | 85 | 76 |
| 0312317 | 黄小全 | 56 | 90 | 75 |
| 0312318 | 李小林 | 76 | 86 | 78 |
+---------+--------+----+----+----+
4 rows in set (0.01 sec)
```

图 12.3　创建视图

运行本实例，通过 MySQL 视图查询学生成绩信息，运行结果如图 12.4 所示，查询结果显示的内容即为视图中所有字段的内容。

视图的应用　学生成绩列表

学号	姓名	语文成绩	外语成绩	数学成绩
0312315	刘小华	88	60	94
0312316	金星星	60	85	76
0312317	黄小全	56	90	75
0312318	李小林	76	86	78

图 12.4　学生成绩列表

（1）创建视图 scoreinfo，通过该视图显示学生成绩信息，该视图的创建代码如下所示。

```
create view scoreinfo as select sno,sname,yw,wy,sx from tb_student,tb_score where tb_student.id=tb_score.sid
```

（2）建立数据库连接文件 conn.php 实现与 MySQL 数据库的连接，并设置数据库字符集为 UTF-8，代码如下。

```
<?php
$conn=new mysqli("localhost","root","root","db_database12");        //连接数据库
$conn->query("set names utf8");                                     //设置编码格式
?>
```

（3）查询视图 scoreinfo 中的内容，并显示查询结果，代码如下。

```
<?php
    include_once("conn.php");                              //包含 conn.php
    $sql=$conn->query("select * from scoreinfo");          //执行查询
    $info=$sql->fetch_array(MYSQLI_ASSOC);                 //获得查询结果集
    if($info==NULL){                                       //判断是否查询到成绩信息
            echo "暂无学生信息";
    }else{
        do{                                                 //通过循环打印学生成绩信息
    ?>
    <tr>
   <td height="20" bgcolor="#FFFFFF"><div align="center"><?php echo $info[sno];?></div></td>
   <td bgcolor="#FFFFFF"><div align="center"><?php echo $info[sname];?></div></td>
```

```
        <td bgcolor="#FFFFFF"><div align="center"><?php echo $info[yw];?></div></td>
        <td bgcolor="#FFFFFF"><div align="center"><?php echo $info[wy];?></div></td>
        <td bgcolor="#FFFFFF"><div align="center"><?php echo $info[sx];?></div></td>
</tr>
        <?php
            }while($info=$sql->fetch_array(MYSQLI_ASSOC));
        }
        ?>
```

12.2.3　创建视图的注意事项

创建视图时需要注意以下几点。

（1）运行创建视图的语句需要用户具有创建视图（create view）的权限，若加了[or replace]时，还需要用户具有删除视图（drop view）的权限。

（2）SELECT 语句不能包含 FROM 子句中的子查询。

（3）SELECT 语句不能引用系统或用户变量。

（4）SELECT 语句不能引用预处理语句参数。

（5）在存储子程序内，定义不能引用子程序参数或局部变量。

（6）在定义中引用的表或视图必须存在。但是，创建了视图后，能够舍弃定义引用的表或视图。要想检查视图定义是否存在这类问题，可使用 CHECK TABLE 语句。

（7）在定义中不能引用 temporary 表，不能创建 temporary 视图。

（8）在视图定义中命名的表必须已存在。

（9）不能将触发程序与视图关联在一起。

（10）在视图定义中允许使用 order by，但是，如果从特定视图进行了选择，而该视图使用了具有自己 order by 的语句，它将被忽略。

12.3　视 图 操 作

12.3.1　查看视图

查看视图是指查看数据库中已存在的视图。查看视图必须要有 SHOW VIEW 的权限。查看视图的方法主要包括 DESCRIBE 语句、SHOW TABLE STATUS 语句、SHOW CREATE VIEW 语句等。本节将主要介绍这几种查看视图的方法。

1．DESCRIBE 语句

DESCRIBE 可以缩写成 DESC，其语法格式如下。

```
DESCRIBE 视图名;
```



下面使用 DESC 语句查询 book_view1 视图中的结构，结果如图 12.5 所示。

```
mysql> DESC book_view1;
+----------+--------------+------+-----+---------+-------+
| Field    | Type         | Null | Key | Default | Extra |
+----------+--------------+------+-----+---------+-------+
| a_sort   | varchar(100) | NO   |     | NULL    |       |
| a_talk   | varchar(100) | NO   |     | NULL    |       |
| a_books  | varchar(100) | NO   |     | NULL    |       |
+----------+--------------+------+-----+---------+-------+
3 rows in set (0.03 sec)
```

图 12.5　使用 DESC 语句查询 book_view1 视图中的结构

结果中显示了字段的名称（Field）、数据类型（Type）、是否为空（Null）、是否为主外键（Key）、默认值（Default）和额外信息（Extra）。

说明

如果只需了解视图中的各个字段的简单信息，可以使用 DESCRIBE 语句。DESCRIBE 语句查看视图的方式与查看普通表的方式是相同的，结果显示的方式也相同。通常情况下，都是使用 DESC 代替 DESCRIBE。

2. SHOW TABLE STATUS 语句

在 MySQL 中，可以使用 SHOW TABLE STATUS 语句查看视图的信息，其语法格式如下。

```
SHOW TABLE STATUS LIKE '视图名';
```

（1）“LIKE”表示后面匹配的是字符串；

（2）“视图名”参数指要查看的视图名称，需要用单引号定义。

例 12.4　下面使用 SHOW TABLE STATUS 语句查看视图 book_view1 中的信息，代码如下。**（实例位置：光盘\TM\sl\12\12.4）**

```
SHOW TABLE STATUS LIKE 'book_view1';
```

执行结果如图 12.6 所示。



```
mysql> SHOW TABLE STATUS LIKE 'book_view1'\G;
*************************** 1. row ***************************
           Name: book_view1
         Engine: NULL
        Version: NULL
     Row_format: NULL
           Rows: NULL
 Avg_row_length: NULL
    Data_length: NULL
Max_data_length: NULL
   Index_length: NULL
      Data_free: NULL
 Auto_increment: NULL
    Create_time: NULL
    Update_time: NULL
     Check_time: NULL
      Collation: NULL
       Checksum: NULL
 Create_options: NULL
        Comment: VIEW
1 row in set (0.00 sec)
```

图 12.6　使用 SHOW TABLE STATUS 语句查看视图 book_view1 中的信息

从执行结果可以看出，存储引擎、数据长度等信息都显示为 NULL，则说明视图为虚拟表，与普通数据表是有区别的。下面使用 SHOW TABLE STATUS 语句来查看 tb_book 表的信息，执行结果如图 12.7 所示。

```
mysql> SHOW TABLE STATUS LIKE 'tb_book'\G;
*************************** 1. row ***************************
           Name: tb_book
         Engine: MyISAM
        Version: 10
     Row_format: Dynamic
           Rows: 14
 Avg_row_length: 204
    Data_length: 2964
Max_data_length: 281474976710655
   Index_length: 2048
      Data_free: 96
 Auto_increment: 35
    Create_time: 2011-06-07 14:00:25
    Update_time: 2011-06-13 10:26:36
     Check_time: NULL
      Collation: utf8_general_ci
       Checksum: NULL
 Create_options:
        Comment:
1 row in set (0.00 sec)
```

图 12.7　使用 SHOW TABLE STATUS 语句来查看 tb_book 表的信息

从上面的结果中可以看出，数据表的信息都已经显示出来了，这就是视图和普通数据表的区别。

3．SHOW CREATE VIEW 语句

在 MySQL 中，SHOW CREATE VIEW 语句可以查看视图的详细定义，其语法格式如下。

```
SHOW CREATE VIEW 视图名
```

例 12.5　下面使用 SHOW CREATE VIEW 语句查看视图 book_view1 的信息，代码如下。**（实例位置：光盘\TM\sl\12\12.5）**

```
SHOW CREATE VIEW book_view1;
```

代码执行结果如图 12.8 所示。

```
mysql> SHOW CREATE VIEW book_view1\G;
*************************** 1. row ***************************
       View: book_view1
Create View: CREATE ALGORITHM=UNDEFINED DEFINER=`root`@`localhost` SQL SECURITY
DEFINER VIEW `book_view1` AS select `tb_book`.`sort` AS `a_sort`,`tb_book`.`talk
` AS `a_talk`,`tb_book`.`books` AS `a_books` from `tb_book`
1 row in set (0.00 sec)
```

图 12.8　使用 SHOW CREATE VIEW 语句查看视图 book_view1 的信息

通过 SHOW CREATE VIEW 语句，可以查看视图的所有信息。

12.3.2　修改视图

修改视图是指修改数据库中已存在的表的定义。当基本表的某些字段发生改变时，可以通过修改



视图来保持视图和基本表之间一致。MySQL 中通过 CREATE OR REPLACE VIEW 语句和 ALTER 语句来修改视图。下面介绍这两种修改视图的方法。

1. CREATE OR REPLACE VIEW

在 MySQL 中，CREATE OR REPLACE VIEW 语句可以用来修改视图。该语句的使用非常灵活。在视图已经存在的情况下，对视图进行修改；视图不存在时，可以创建视图。CREATE OR REPLACE VIEW 语句的语法如下。

```
CREATE OR REPLACE [ALGORITHM={UNDEFINED | MERGE | TEMPTABLE}]
VIEW 视图[(属性清单)]
AS SELECT 语句
[WITH [CASCADED | LOCAL] CHECK OPTION];
```

例 12.6　下面使用 CREATE OR REPLACE VIEW 语句将视图 book_view1 的字段修改为 a_sort 和 a_book，执行结果如图 12.9 所示。**（实例位置：光盘\TM\sl\12\12.6）**

```
mysql> CREATE OR REPLACE ALGORITHM=TEMPTABLE
    -> VIEW book_view1(a_sort,a_book)
    -> AS SELECT sort,books FROM tb_book;
Query OK, 0 rows affected (0.00 sec)
```

图 12.9　使用 CREATE OR REPLACE VIEW 语句修改视图

使用 DESC 语句查询 book_view1 视图，结果如图 12.10 所示。

```
mysql> DESC book_view1;
+--------+--------------+------+-----+---------+-------+
| Field  | Type         | Null | Key | Default | Extra |
+--------+--------------+------+-----+---------+-------+
| a_sort | varchar(100) | NO   |     | NULL    |       |
| a_book | varchar(100) | NO   |     | NULL    |       |
+--------+--------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

图 12.10　使用 DESC 语句查询 book_view1

从上面的结果中可以看出，修改后的 book_view1 中只有两个字段。

2. ALTER

ALTER VIEW 语句改变了视图的定义，包括被索引视图，但不影响所依赖的存储过程或触发器。该语句与 CREATE VIEW 语句有着同样的限制，如果删除并重建了一个视图，就必须重新为它分配权限。

ALTER VIEW 语句的语法如下。

```
alter view [algorithm={merge | temptable | undefined} ]view view_name [(column_list)] as select_statement[with [cascaded | local] check option]
```

（1）algorithm：该参数已经在创建视图中做了介绍，这里不再赘述。

（2）view_name：视图的名称。

（3）select_statement：SQL 语句用于限定视图。

注意

在创建视图时，在使用了 WITH CHECK OPTION，WITH ENCRYPTION,WITH SCHEMABING 或 VIEW_METADATA 选项时，如果想保留这些选项提供的功能，必须在 ALTER VIEW 语句中将它们包括进去。

例 12.7 下面将对 book_view1 视图进行修改，将原有的 a_sort 和 a_book 两个属性更改为 a_sort 一个属性。在更改前，先来查看一下 book_view1 视图此时包含的属性，结果如图 12.11 所示。**（实例位置：光盘\TM\sl\12\12.7）**

```
mysql> DESC book_view1;
+--------+--------------+------+-----+---------+-------+
| Field  | Type         | Null | Key | Default | Extra |
+--------+--------------+------+-----+---------+-------+
| a_sort | varchar(100) | NO   |     | NULL    |       |
| a_book | varchar(100) | NO   |     | NULL    |       |
+--------+--------------+------+-----+---------+-------+
2 rows in set (0.00 sec)
```

图 12.11 查看 book_view1 视图的属性

从结果中可以看出，此时的 book_view1 视图中包含两个属性，下面对视图进行修改，结果如图 12.12 所示。

```
mysql> ALTER VIEW book_view1(a_sort)
    -> AS SELECT sort
    -> FROM tb_book
    -> WITH CHECK OPTION;
Query OK, 0 rows affected (0.00 sec)
```

图 12.12 修改视图属性

结果显示修改成功，下面再来查看一下修改后的视图属性，结果如图 12.13 所示。

```
mysql> DESC book_view1;
+--------+--------------+------+-----+---------+-------+
| Field  | Type         | Null | Key | Default | Extra |
+--------+--------------+------+-----+---------+-------+
| a_sort | varchar(100) | NO   |     | NULL    |       |
+--------+--------------+------+-----+---------+-------+
1 row in set (0.00 sec)
```

图 12.13 查看修改后的视图属性

结果显示，此时视图中只包含一个 a_sort 属性。

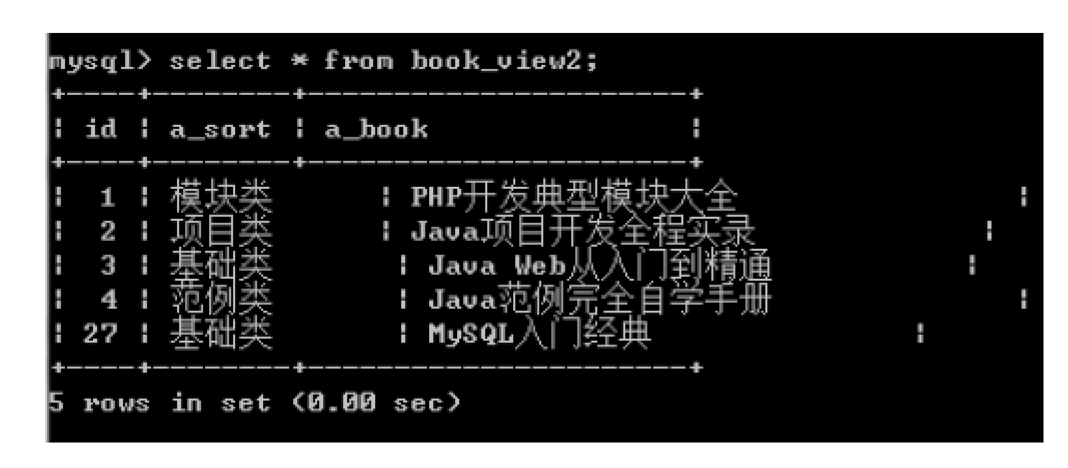
12.3.3 更新视图

对视图的更新其实就是对表的更新，更新视图是指通过视图来插入（INSERT）、更新（UPDATE）和删除（DELETE）表中的数据。因为视图是一个虚拟表，其中没有数据。通过视图更新时，都是转换到基本表来更新。更新视图时，只能更新权限范围内的数据，超出了范围，就不能更新。本节讲解更新视图的方法和更新视图的限制。



1．更新视图

例 12.8　下面对 book_view2 视图中的数据进行更新，先来查看 book_view2 视图中的数据，如图 12.14 所示。**（实例位置：光盘\TM\sl\12\12.8）**



```
mysql> select * from book_view2;
+----+--------+------------------------+
| id | a_sort | a_book                 |
+----+--------+------------------------+
|  1 | 模块类 | PHP开发典型模块大全    |
|  2 | 项目类 | Java项目开发全程实录   |
|  3 | 基础类 | Java Web从入门到精通   |
|  4 | 范例类 | Java范例完全自学手册   |
| 27 | 基础类 | MySQL入门经典          |
+----+--------+------------------------+
5 rows in set (0.00 sec)
```

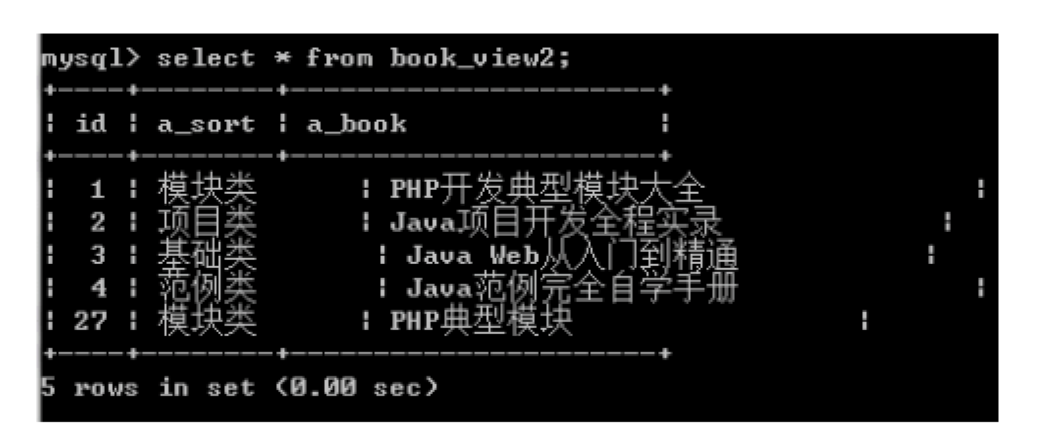
图 12.14　查看 book_view2 视图中的数据

下面更新视图中的第 27 条记录，a_sort 的值为“模块类”，a_book 的值为“PHP 典型模块”，更新语句结果如图 12.15 所示。

```
mysql> UPDATE book_view2 SET a_sort='模块类',a_book='PHP典型模块' WHERE id=27;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0
```

图 12.15　更新视图中的数据

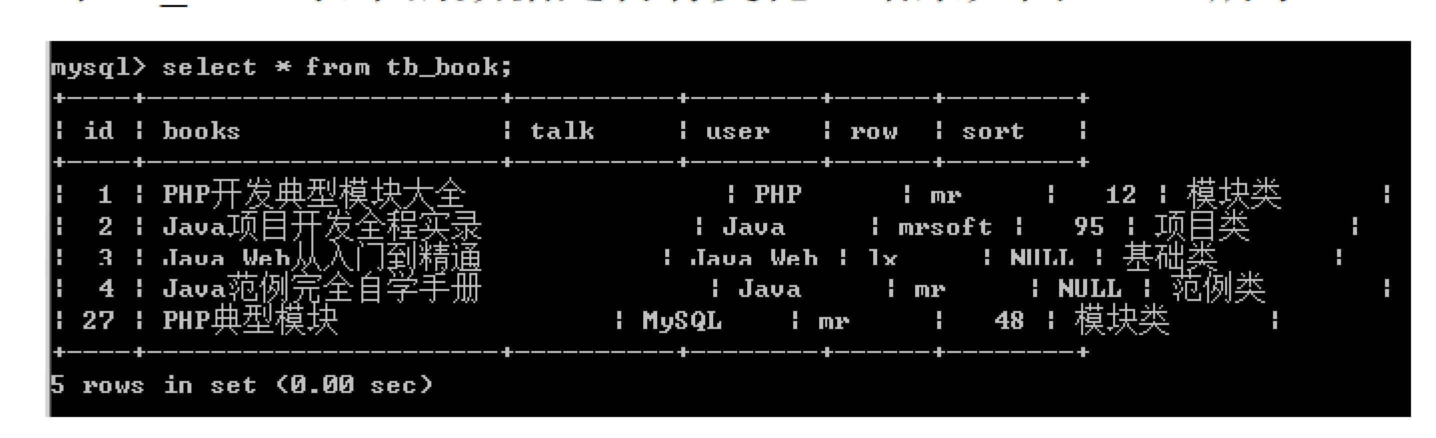
结果显示更新成功，下面再来查看一下 book_view2 视图中的数据是否有变化，结果如图 12.16 所示。



```
mysql> select * from book_view2;
+----+--------+------------------------+
| id | a_sort | a_book                 |
+----+--------+------------------------+
|  1 | 模块类 | PHP开发典型模块大全    |
|  2 | 项目类 | Java项目开发全程实录   |
|  3 | 基础类 | Java Web从入门到精通   |
|  4 | 范例类 | Java范例完全自学手册   |
| 27 | 模块类 | PHP典型模块            |
+----+--------+------------------------+
5 rows in set (0.00 sec)
```

图 12.16　查看更新后视图中的数据

下面再来查看一下 tb_book 表中的数据是否有变化，结果如图 12.17 所示。



```
mysql> select * from tb_book;
+----+----------------------+----------+--------+------+--------+
| id | books                | talk     | user   | row  | sort   |
+----+----------------------+----------+--------+------+--------+
|  1 | PHP开发典型模块大全  | PHP      | mr     |   12 | 模块类 |
|  2 | Java项目开发全程实录 | Java     | mrsoft |   95 | 项目类 |
|  3 | Java Web从入门到精通 | Java Web | lx     | NULL | 基础类 |
|  4 | Java范例完全自学手册 | Java     | mr     | NULL | 范例类 |
| 27 | PHP典型模块          | MySQL    | mr     |   48 | 模块类 |
+----+----------------------+----------+--------+------+--------+
5 rows in set (0.00 sec)
```

图 12.17　查看 tb_book 表中的数据

从上面的结果可以看出，对视图的更新其实就是对基本表的更新。

2．更新视图的限制

并不是所有的视图都可以更新，以下几种情况是不能更新视图的。

（1）视图中包含 COUNT()、SUM()、MAX()和 MIN()等函数。例如：

```
CREATE VIEW book_view1(a_sort,a_book)
AS SELECT sort,books, COUNT(name) FROM tb_book;
```

（2）视图中包含 UNION、UNION ALL、DISTINCT、GROUP BY 和 HAVIG 等关键字。例如：

```
CREATE VIEW book_view1(a_sort,a_book)
AS SELECT sort,books, FROM tb_book GROUP BY id;
```

（3）常量视图。例如：

```
CREATE VIEW book_view1
AS SELECT 'Aric' as a_book;
```

（4）视图中的 SELECT 中包含子查询。例如：

```
CREATE VIEW book_view1(a_sort)
AS SELECT (SELECT name FROM tb_book);
```

（5）由不可更新的视图导出的视图。例如：

```
CREATE VIEW book_view1
AS SELECT * FROM book_view2;
```

（6）创建视图时，ALGORITHM 为 TEMPTABLE 类型。例如：

```
CREATE ALGORITHM=TEMPTABLE
VIEW book_view1
AS SELECT * FROM tb_book;
```

（7）视图对应的表上存在没有默认值的列，而且该列没有包含在视图里。例如，表中包含的 name 字段没有默认值，但是视图中不包括该字段，那么这个视图是不能更新的。因为，在更新视图时，这个没有默认值的记录将没有值插入，也没有 NULL 值插入。数据库系统是不会允许这样的情况出现的，其会阻止这个视图更新。

上面的几种情况其实就是一种情况，即视图的数据和基本表的数据不一样了。

注意

视图中虽然可以更新数据，但是有很多的限制。一般情况下，最好将视图作为查询数据的虚拟表，而不要通过视图更新数据。因为，使用视图更新数据时，如果没有全面考虑在视图中更新数据的限制，可能会造成数据更新失败。

12.3.4　删除视图

删除视图是指删除数据库中已存在的视图。删除视图时，只能删除视图的定义，不会删除数据。MySQL 中，使用 DROP VIEW 语句来删除视图。但是，用户必须拥有 DROP 权限。本节将介绍删除视图的方法。



DROP VIEW 语句的语法如下。

```
DROP VIEW IF EXISTS <视图名> [RESTRICT | CASCADE]
```

（1）IF EXISTS 参数指判断视图是否存在，如果存在则执行，不存在则不执行。

（2）“视图名”列表参数表示要删除的视图的名称和列表，各个视图名称之间用逗号隔开。

该语句从数据字典中删除指定的视图定义；如果该视图导出了其他视图，则使用 CASCADE 级联删除，或者先显式删除导出的视图，再删除该视图；删除基表时，由该基表导出的所有视图定义都必须显式删除。

例 12.9　下面删除前面实例中一直使用的 book_view1 视图，执行语句如下。**（实例位置：光盘\TM\sl\12\12.9）**

```
DROP VIEW IF EXISTS book_view1;
```

执行结果如图 12.18 所示。

```
mysql> DROP VIEW IF EXISTS book_view1;
Query OK, 0 rows affected (0.00 sec)
```

图 12.18　删除视图

执行结果显示删除成功。下面验证一下视图是否真正被删除，执行 SHOW CREATE VIEW 语句查看，执行结果如图 12.19 所示。

```
mysql> SHOW CREATE VIEW book_view1;
ERROR 1146 (42S02): Table 'db_book.book_view1' doesn't exist
```

图 12.19　查看视图是否删除成功

结果显示，视图 book_view1 已经不存在了，说明 DROP VIEW 语句删除视图成功。

12.4　小　　结

本章对 MySQL 数据库的视图的含义和作用进行了详细讲解，并且讲解了创建视图、修改视图和删除视图的方法。创建视图和修改视图是本章的重点内容，并且需要在计算机上实际操作。读者在创建视图和修改视图后，一定要查看视图的结构，以确保创建和修改的操作正确。更新视图是本章的一个难点，因为实际中存在一些造成视图不能更新的因素，希望读者在练习中认真分析。

12.5　实践与练习

1．在 department 表上创建一个简单的视图，视图名称为 department_view1。**（答案位置：光盘\TM\sl\12\12.10）**

2．使用 DESCRIBE 语句查询视图的结构。**（答案位置：光盘\TM\sl\12\12.11）**

第 3 篇

高级应用

本篇介绍数据完整性约束、存储过程与存储函数、触发器、事务的应用、事件、备份与恢复、MySQL 性能优化、权限管理及安全控制、PHP 管理 MySQL 数据库中的数据等。学习完这一部分，能够掌握如何进行数据的导入与导出操作，以及使用存储过程、触发器、事务、事件等。通过这些内容不仅可以优化查询，还可以提高数据访问速度；更好地维护 MySQL 的权限及其安全。另外，还介绍了应用 PHP 管理 MySQL 数据库中的数据，对于想要使用 PHP 开发的读者非常实用。

第13章

数据完整性约束

数据完整性是指数据的正确性和相容性，是为了防止数据库中存在不符合语义的数据，即防止数据库中存在不正确的数据。在MySQL中提供了多种完整性约束，它们作为数据库关系模式定义的一部分，可以通过CREATE TABLE或ALTER TABLE语句来定义。一旦定义了完整性约束，MySQL服务器会随时检测处于更新状态的数据库内容是否符合相关的完整性约束，从而保证数据的一致性与正确性。这样，既能有效地防止对数据库的意外破坏，又能提高完整性检测的效率，还能减轻数据库编程人员的工作负担。本章中将对数据完整性约束进行详细介绍。

通过阅读本章，读者可以：

- 掌握定义完整性约束的方法
- 掌握命名完整性约束的方法
- 掌握更新完整性约束的方法

13.1　定义完整性约束

关系模型的完整性规则是对关系的某种约束条件。在关系模型中，提供了实体完整性、参照完整性和用户定义完整性 3 项规则。下面将分别介绍 MySQL 中对数据库完整性 3 项规则的设置和实现方式。

13.1.1　实体完整性

实体（Entity）是一个数据对象，是指客观存在并可以相互区分的事物，如一个教师、一个学生或一个雇员等。一个实体在数据库中表现为表中的一条记录。通常情况下，它必须遵守实体完整性规则。

实体完整性规则（Entity Integrity Rule）是指关系的主属性，即主码（主键）的组成不能为空，也就是关系的主属性不能是空值（NULL）。关系对应于现实世界中的实体集，而现实世界中的实体是可区分的，即说明每个实例具有唯一性标识。在关系模型中，是使用主码（主键）作为唯一性标识的，若假设主码（主键）取空值，则说明这个实体不可标识，即不可区分，这个假设显然不正确，与现实世界应用环境相矛盾，因此不能存在这样的无标识实体，从而在关系模型中引入实体完整性约束。例如，学生关系（学号、姓名、性别）中，“学号”为主码（主键），则“学号”这个属性不能为空值，否则就违反了实体完整性规则。

在 MySQL 中，实体完整性是通过主键约束和候选键约束来实现的。

1．主键约束

主键可以是表中的某一列，也可以是表中多个列所构成的一个组合；其中，由多个列组合而成的主键也称为复合主键。在 MySQL 中，主键列必须遵守以下规则。

（1）每一个表只能定义一个主键。

（2）唯一性原则。主键的值，也称键值，必须能够唯一标识表中的每一行记录，且不能为 NULL。也就是说一张表中两个不同的行在主键上不能具有相同的值。

（3）最小化规则。复合主键不能包含不必要的多余列。也就是说，当从一个复合主键中删除一列后，如果剩下的列构成的主键仍能满足唯一性原则，那么这个复合主键是不正确的。

（4）一个列名在复合主键的列表中只能出现一次。

在 MySQL 中，可以在 CREATE TABLE 或者 ALTER TABLE 语句中，使用 PRIMARY KEY 子句来创建主键约束，其实现方式有以下两种。

1）作为列的完整性约束

在表的某个列的属性定义时，加上 PRIMARY KEY 关键字实现。

例 13.1　在创建用户信息表 tb_user 时，将 id 字段设置为主键，代码如下。**（实例位置：光盘\TM\sl\13\13.1）**

```
create table tb_user(
```

```
id int auto_increment primary key,
user varchar(30) not null,
password varchar(30) not null,
createtime datetime);
```

运行上述代码，其结果如图 13.1 所示。

2）作为表的完整性约束

在表的所有列的属性定义后，加上 PRIMARY KEY(index_col_name,…)子句实现。

例 13.2 创建学生信息表 tb_student 时，将学号（id）和所在班级号（classid）字段设置为主键，代码如下。**（实例位置：光盘\TM\sl\13\13.2）**

```
create table tb_student (
id int auto_increment,
name varchar(30) not null,
sex varchar(2),
classid int not null,
birthday date,
PRIMARY KEY (id,classid)
);
```

运行上述代码，其结果如图 13.2 所示。

```
mysql> use db_database13;
Database changed
mysql> create table tb_user(
    -> id int auto_increment primary key,
    -> user varchar(30) not null,
    -> password varchar(30) not null,
    -> createtime datetime);
Query OK, 0 rows affected (0.56 sec)

mysql> _
```

图 13.1 将 id 字段设置为主键

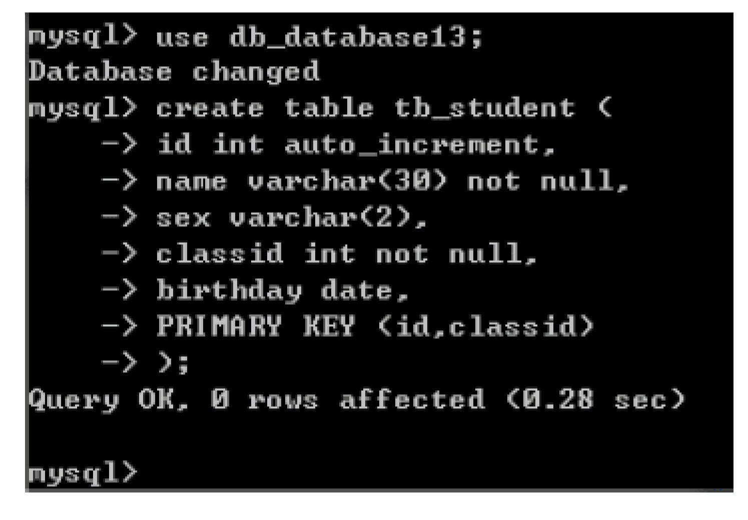


图 13.2 将 id 字段和 classid 字段设置为主键

说明

如果主键仅由表中的某一列所构成，那么以上两种方法均可以定义主键约束；如果主键由表中多个列所构成，那么只能用第二种方法定义主键约束。另外，定义主键约束后，MySQL 会自动为主键创建一个唯一索引，默认名为 PRIMARY，也可以修改为其他名称。

2．候选键约束

如果一个属性集能唯一标识元组，且又不含有多余的属性，那么这个属性集称为关系的候选键。例如，在包含学号、姓名、性别、年龄、院系、班级等列的“学生信息表”中，“学号”能够标识一名学生，因此，它可以作为候选键，而如果规定，不允许有同名的学生，那么姓名也可以作为候选键。

候选键可以是表中的某一列，也可以是表中多个列所构成的一个组合。任何时候，候选键的值必须是唯一的，且不能为空（NULL）。候选键可以在 CREATE TABLE 或者 ALTER TABLE 语句中使用关键字 UNIQUE 来定义，其实现方法与主键约束类似，也是可作为列的完整性约束或者表的完整性约

束两种方式。

在 MySQL 中，候选键与主键之间存在以下两点区别。

（1）一个表只能创建一个主键，但可以定义若干个候选键。

（2）定义主键约束时，系统会自动创建 PRIMARY KEY 索引，而定义候选键约束时，系统会自动创建 UNIQUE 索引。

例 13.3　在创建用户信息表 tb_user1 时，将 id 字段和 user 字段设置为候选键，代码如下。**（实例位置：光盘\TM\sl\13\13.3）**

```
create table tb_user1(
id int auto_increment UNIQUE,
user varchar(30) not null UNIQUE,
password varchar(30) not null,
createtime TIMESTAMP default CURRENT_TIMESTAMP);
```

运行上述代码，其结果如图 13.3 所示。

```
mysql> use db_database13;
Database changed
mysql> create table tb_user1(
    -> id int auto_increment UNIQUE,
    -> user varchar(30) not null UNIQUE,
    -> password varchar(30) not null,
    -> createtime TIMESTAMP default CURRENT_TIMESTAMP);
Query OK, 0 rows affected (0.46 sec)

mysql> _
```

图 13.3　将 id 字段和 user 字段设置为候选键

13.1.2　参照完整性

现实世界中的实体之间往往存在着某种联系，在关系模型中，实体及实体间的联系都是用关系来描述的，那么自然就存在着关系与关系间的引用。例如，学生实体和班级实体可以分别用下面的关系表示，其中，主码（主键）用下划线标识。

学生（学生证号，姓名，性别，生日，班级编号，备注）

班级（班级编号，班级名称，备注）

在这两个关系之间存在着属性的引用，即“学生”关系引用了“班级”关系中的主码（主键）“班级编号”。 在两个实体间，“班级编号”是“班级”关系的主码（主键），也是“学生”关系的外部码（外键）。显然，“学生”关系中的“班级编号”的值必须是确实存在的班级的“班级编号”，即“班级”关系中的该班级的记录。也就是说，“学生”关系中某个属性的取值需要参照“班级”关系的属性和值。

参照完整性规则（Referential Integrity Rule）就是定义外码（外键）和主码（主键）之间的引用规则，它是对关系间引用数据的一种限制。

参照完整性的定义为：若属性（或属性组）F 是基本关系 R 的外码，它与基本关系 S 的主码 K 相对应，则对于 R 中每个元组在 F 上的值只允许两种可能，即要么取空值（F 的每个属性值均为空值），要么等于 S 中某个元组的主码值。其中，关系 R 与 S 可以是不同的关系，也可以是同一关系，而 F 与 K 是

定义在同一个域中。例如，在“学生”关系中每个学生的“班级编号”一项，要么取空值，表示该学生还没有分配班级；要么取值必须与“班级”关系中的某个元组的“班级编号”相同，表示这个学生分配到某个班级学习。这就是参照完整性。如果“学生”关系中，某个学生的“班级编号”取值不能与“班级”关系中任何一个元组的“班级编号”值相同，表示这个学生被分配到不属于所在学校的班级学习，这与实际应用环境不相符，显然是错误的，这就需要在关系模型中定义参照完整性进行约束。

与实体完整性一样，参照完整性也是由系统自动支持的，即在建立关系（表）时，只要定义了“谁是主码”“谁参照于认证”，系统将自动进行此类完整性的检查。在 MySQL 中，参照完整性可以通过在创建表（CREATE TABLE）或者修改表（ALTER TABLE）时定义一个外键声明来实现。

MySQL 有两种常用的引擎类型（MyISAM 和 InnoDB），目前，只有 InnoDB 引擎类型支持外键约束。InnoDB 引擎类型中声明外键的基本语法格式如下。

```
[CONSTRAINT [SYMBOL]]
FOREIGN KEY (index_col_name,…)   reference_definition
```

reference_definition 主要用于定义外键所参照的表、列、参照动作的声明和实施策略等 4 部分内容。它的基本语法格式如下。

```
REFERENCES tbl_name [(index_col_name,…)]
                  [MATCH FULL | MATCH PARTIAL | MATCH SIMPLE]
                  [ON DELETE reference_option]
                  [ON UPDATE reference_option]
```

index_col_name 的语法格式如下。

```
col_name [(length)] [ASC | DESC]
```

reference_option 的语法格式如下。

```
RESTRICT | CASCADE | SET NULL | NO ACTION
```

参数说明如下。

（1）index_col_name：用于指定被设置为外键的列。

（2）tbl_name：用于指定外键所参照的表名。这个表称为参照表（或父表），而外键所在的表称作参照表（或子表）。

（3）col_name：用于指定被参照的列名。外键可以引用被参照表中的主键或候选键，也可以引用被参照表中某些列的一个组合，但这个组合不能是被参照表中随机的一组列，必须保存该组合的取值在被参照表中是唯一的。外键中的所有列值在被参照表的列中必须全部存在，也就是通过外键来对参照表某些列（外键）的取值进行限定与约束。

（4）ON DELETE | ON UPDATE：指定参照动作相关的 SQL 语句。可为每个外键指定对应于 DELETE 语句和 UPDATE 语句的参照动作。

（5）reference_option：指定参照完整性约束的实现策略。其中，当没有明确指定参照完整性的实现策略时，两个参照动作会默认使用 RESTRICT。具体的策略可选值如表 13.1 所示。



表 13.1 策略可选值

可 选 值	说 明
RESTRICT	限制策略：当要删除或更新被参照表中被参照列上，并在外键中出现的值时，系统拒绝对被参照表的删除或更新操作
CASCADE	级联策略：从被参照表中删除或更新记录行时，自动删除或更新参照表匹配的记录行
SET NULL	置空策略：当从被参照表中删除或更新记录行时，设置参照表中与之对应的外键列的值为 NULL。这个策略需要被参照表中的外键列没有声明限定词 NOT NULL
NO ACTION	不采取实施策略：当一个相关的外键值在被参照表中时，删除或更新被参照表中键值的动作不被允许。该策略的动作语言与 RESTRICT 相同

例 13.4 创建学生信息表 tb_student1，并为其设置参照完整性约束（拒绝删除或更新被参照表中被参照列上的外键值），即将 classid 字段设置为外键，代码如下。**（实例位置：光盘\TM\sl\13\ 13.4）**

```
create table tb_student1 (
id int auto_increment,
name varchar(30) not null,
sex varchar(2),
classid int not null,
birthday date,
remark varchar(100),
primary key (id),
FOREIGN KEY (classid)
REFERENCES tb_class(id)
ON DELETE RESTRICT
ON UPDATE RESTRICT
);
```

运行上述代码，其结果如图 13.4 所示。



图 13.4 将 classid 字段设置为外键

注意

要设置为主外键关系的两张数据表必须具有相同的存储引擎，如都是 InnoDB，并且相关联的两个字段的类型必须一致。

设置外键时，通常需要遵守以下规则。

（1）被参照表必须是已经存在的，或者是当前正在创建的表。如果是当前正在创建的表，也就是说，被参照表与参照表是同一个表，这样的表称为自参照表（Self-referencing Table），这种结构称为自参照完整性（Self-referential Integrity）。

（2）必须为被参照表定义主键。

（3）必须在被参照表名后面指定列名或列名的组合。这个列或列组合必须是这个被参照表的主键或候选键。

（4）外键中列的数目必须和被参照表中的列的数据相同。

（5）外键中列的数据类型必须和被参照表的主键（或候选键）中的对应列的数据类型相同。

（6）尽管主键是不能够包含空值的，但允许在外键中出现一个空值。这意味着，只要外键的每个非空值出现在指定的主键中，这个外键的内容就是正确的。

13.1.3　用户定义完整性

用户定义完整性规则（User-defined Integrity Rule）是针对某一应用环境的完整性约束条件，它反映了某一具体应用所涉及的数据应满足的要求。关系模型提供定义和检验这类完整性规则的机制，其目的是由系统来统一处理，而不再由应用程序来完成这项工作。在实际系统中，这类完整性规则一般是在建立数据表的同时进行定义，应用编程人员不需要再做考虑，如果某些约束条件没有建立在库表一级，则应用编程人员应在各模块的具体编程中通过程序进行检查和控制。

MySQL 支持非空约束、CHECK 约束和触发器 3 种用户自定义完整性约束。其中，触发器将在第 15 章进行详细介绍。这里主要介绍非空约束和 CHECK 约束。

1．非空约束

在 MySQL 中，非空约束可以通过在 CREATE TABLE 或 ALTER TABLE 语句中，某个列定义后面加上关键字 NOT NULL 来定义，用来约束该列的取值不能为空。

例 13.5　创建班级信息表 tb_class1，并为其 name 字段添加非空约束，代码如下。**（实例位置：光盘\TM\sl\13\13.5）**

```
CREATE TABLE tb_class1 (
	id int(11) NOT NULL AUTO_INCREMENT,
	name varchar(45) NOT NULL,
	remark varchar(100) DEFAULT NULL,
	PRIMARY KEY (`id`)
);
```

运行上述代码，其结果如图 13.5 所示。

```
mysql> use db_database13;
Database changed
mysql> CREATE TABLE tb_class1 (
    ->  id int(11) NOT NULL AUTO_INCREMENT,
    ->  name varchar(45) NOT NULL,
    ->  remark varchar(100) DEFAULT NULL,
    ->  PRIMARY KEY (`id`)
    -> );
Query OK, 0 rows affected (0.40 sec)

mysql>
```

图 13.5　为 name 字段添加非空约束



2．CHECK 约束

与非空约束一样，CHECK 约束也可以通过在 CREATE TABLE 或 ALTER TABLE 语句中，根据用户的实际完整性要求来定义。它可以分别对列或表实施 CHECK 约束，其中使用的语法如下。

```
CHECK(expr)
```

其中，expr 是一个 SQL 表达式，用于指定需要检查的限定条件。在更新表数据时，MySQL 会检查更新后的数据行是否满足 CHECK 约束中的限定条件。该限定条件可以是简单的表达式，也可以是复杂的表达式（如子查询）。

下面将分别介绍如何对列和表实施 CHECK 约束。

1）对列实施 CHECK 约束

将 CHECK 子句置于表的某个列的定义之后就是对列实施 CHECK 约束。下面将通过一个具体的实例来说明如何对列实施 CHECK 约束。

例 13.6　创建学生信息表 tb_student2，限制其 age 字段的值只能在 7~18 之间（不包括 18）的数，代码如下。**（实例位置：光盘\TM\sl\13\13.6）**

```
create table tb_student2 (
id int auto_increment,
name varchar(30) not null,
sex varchar(2),
age int not null CHECK(age>6 and age<18),
remark varchar(100),
primary key (id)
);
```

运行上述代码，其结果如图 13.6 所示。

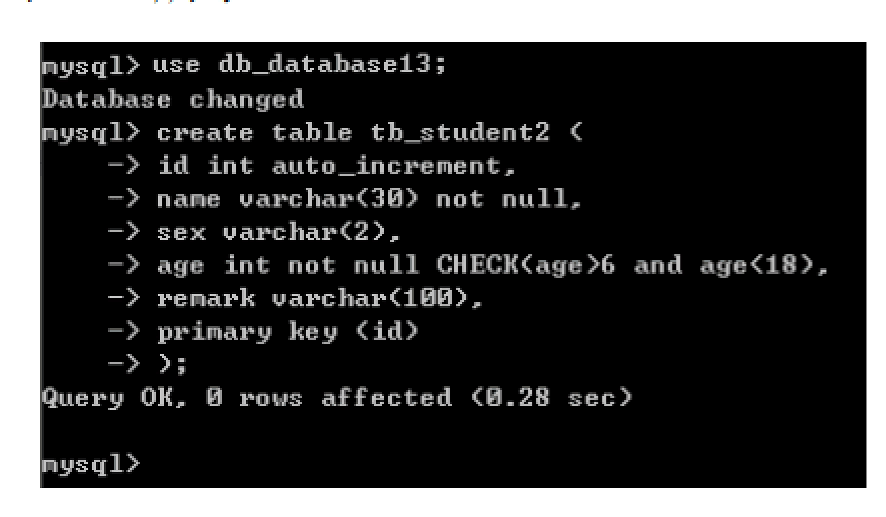


图 13.6　对列实施 CHECK 约束



目前的 MySQL 版本只是对 CHECK 约束进行了分析处理，但会被直接忽略，并不会报错。

2）对表实施 CHECK 约束

将 CHECK 子句置于表中所有列的定义以及主键约束和外键定义之后就是对表实施 CHECK 约束。下面将通过一个具体的实例来说明如何对表实施 CHECK 约束。

例 13.7　创建学生信息表 tb_student3，限制其 classid 字段的值只能是 tb_class 表中 id 字段的某一个 id 值，代码如下。**（实例位置：光盘\TM\sl\13\13.7）**

```
create table tb_student3 (
id int auto_increment,
name varchar(30) not null,
sex varchar(2),
classid int not null,
birthday date,
remark varchar(100),
primary key (id),
CHECK(classid IN (SELECT id FROM tb_class))
);
```

运行上述代码，其结果如图 13.7 所示。

```
mysql> use db_database13;
Database changed
mysql> create table tb_student3 (
    -> id int auto_increment,
    -> name varchar(30) not null,
    -> sex varchar(2),
    -> classid int not null,
    -> birthday date,
    -> remark varchar(100),
    -> primary key (id),
    -> CHECK(classid IN (SELECT id FROM tb_class))
    -> );
Query OK, 0 rows affected (0.39 sec)

mysql> _
```

图 13.7　对表实施 CHECK 约束

13.2　命名完整性约束

在 MySQL 中，也可以对完整性约束进行添加、修改和删除等操作。其中，为了删除和修改完整性约束，需要在定义约束的同时对其进行命名。命名完整性约束的方式是在各种完整性约束的定义说明之前加上 CONSTRAINT 子句实现的。CONSTRAINT 子句的语法格式如下。

```
CONSTRAINT <symbol>
     [PRIMAR KEY 短语 |FOREIGN KEY 短语 |CHECK 短语]
```

参数说明如下。

（1）symbol：用于指定约束名称。这个名字是在完整性约束说明的前面被定义，在数据库里必须是唯一的。如果在创建时没有指定约束的名字，则 MySQL 将自动创建一个约束名字。

（2）PRIMAR KEY 短语：主键约束。

（3）FOREIGN KEY 短语：参照完整性约束。

（4）CHECK 短语：CHECK 约束。

说明

在 MySQL 中，主键约束名称只能是 PRIMARY。



例如，对雇员表添加主键约束，并为其命名为 PRIMARY，可以使用下面的代码。

```
ALTER TABLE 雇员表 ADD CONSTRAINT PRIMARY
PRIMARY KEY (雇员编号)
```

例 13.8　修改例 13.4 的代码，重新创建学生信息表 tb_student1，命名为 tb_student1a，并为其参照完整性约束命名，代码如下。（**实例位置：光盘\TM\sl\13\13.8**）

```
create table tb_student1a (
id int auto_increment PRIMARY KEY,
name varchar(30) not null,
sex varchar(2),
classid int not null,
birthday date,
remark varchar(100),
CONSTRAINT fk_classid FOREIGN KEY (classid)
REFERENCES tb_class(id)
ON DELETE RESTRICT
ON UPDATE RESTRICT
);
```

运行上述代码，其结果如图 13.8 所示。

```
mysql> use db_database13;
Database changed
mysql> create table tb_student1a (
    -> id int auto_increment PRIMARY KEY,
    -> name varchar(30) not null,
    -> sex varchar(2),
    -> classid int not null,
    -> birthday date,
    -> remark varchar(100),
    -> CONSTRAINT fk_classid FOREIGN KEY (classid)
    -> REFERENCES tb_class(id)
    -> ON DELETE RESTRICT
    -> ON UPDATE RESTRICT
    -> );
Query OK, 0 rows affected (0.43 sec)

mysql> _
```

图 13.8　命名完整性约束



说明

在定义完整性约束时，应该尽可能为其指定名字，以便在需要对完整性约束进行修改或删除时，可以很容易地找到它们。



注意

只能给基于表的完整性约束指定名字，无法给基于列的完整性约束指定名字。

例 13.9　在创建表时添加命名外键完整性约束。（**实例位置：光盘\TM\sl\13\13.9**）

在本实例中，首先创建一个图书类别信息表，然后再创建一个图书信息表，并为图书信息表设置命名外键约束，实现删除参照表中的数据时，级联删除图书信息表中相关类别的图书信息，具体步骤如下。

（1）创建名称为 tb_type 的图书类别信息表，具体代码如下。

```
CREATE TABLE tb_type (
      id int(11) NOT NULL AUTO_INCREMENT,
      name varchar(45) DEFAULT NULL,
      remark varchar(100) DEFAULT NULL,

      PRIMARY KEY (`id`)
);
```

（2）创建不添加任何外键的教材信息表 tb_book，代码如下。

```
Create table tb_book(id int(11) not null primary key auto_increment,
name varchar(20) not null,
publishingho varchar(20) not null,
author varchar(20),
typeid int(11),
CONSTRAINT fk_typeid
FOREIGN KEY (typeid)
REFERENCES tb_type(id)
ON DELETE CASCADE
ON UPDATE CASCADE
);
```

运行结果如图 13.9 所示。

```
mysql> use db_database13;
Database changed
mysql> CREATE TABLE tb_type (
    ->  id int(11) NOT NULL AUTO_INCREMENT,
    ->  name varchar(45) DEFAULT NULL,
    ->  remark varchar(100) DEFAULT NULL,
    ->
    ->  PRIMARY KEY (`id`)
    -> );
Query OK, 0 rows affected (0.38 sec)

mysql> Create table tb_book(id int(11) not null primary key auto_increment,
    -> name varchar(20) not null,
    -> publishingho varchar(20) not null,
    -> author varchar(20),
    -> typeid int(11),
    -> CONSTRAINT fk_typeid
    -> FOREIGN KEY (typeid)
    -> REFERENCES tb_type(id)
    -> ON DELETE CASCADE
    -> ON UPDATE CASCADE
    -> );
Query OK, 0 rows affected (0.34 sec)

mysql> _
```

图 13.9　在创建表时添加命名外键完整性约束

13.3　更新完整性约束

对各种约束命名后，就可以使用 ALTER TABLE 语句来更新或删除与列或表有关的各种约束。下面将分别进行介绍。

13.3.1　删除完整性约束

在 MySQL 中，使用 ALTER TABLE 语句，可以独立地删除完整性约束，而不会删除表本身。如果使用 DROP TABLE 语句删除一个表，那么这个表中的所有完整性约束也会自动被删除。删除完整性约束需要在 ALTER TABLE 语句中使用 DROP 关键字来实现，具体的语法格式如下。

```
DROP [FOREIGN KEY| INDEX| <symbol>] |[PRIMARY KEY]
```

参数说明如下。

（1）FOREIGN KEY：用于删除外键约束。

（2）PRIMARY KEY：用于删除主键约束。需要注意的是，在删除主键时，必须再创建一个主键，否则不能删除成功。

（3）INDEX：用于删除候选键约束。

（4）symbol：要删除的约束名称。

例 13.10　要删除例 13.8 中的名称为 fk_classid 的外键约束，可以使用下面的代码。**（实例位置：光盘\TM\sl\13\13.10）**

```
ALTER TABLE tb_student1a DROP FOREIGN KEY fk_classid;
```

运行上述代码，其结果如图 13.10 所示。

```
mysql> ALTER TABLE tb_student1a DROP FOREIGN KEY fk_classid;
Query OK, 0 rows affected (0.08 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql>
```

图 13.10　删除名称为 fk_classid 的外键约束

13.3.2　修改完整性约束

在 MySQL 中，完整性约束不能直接被修改，若要修改只能使用 ALTER TABLE 语句先删除除该约束，然后再增加一个与该约束同名的新约束。由于删除完整性约束的语法在 13.3.1 节已经介绍了，这里只给出在 ALTER TABLE 语句中添加完整性约束的语法格式，具体语法格式如下。

```
ADD CONSTRAINT <symbol> 各种约束
```

参数说明如下。

（1）symbol：为要添加的约束指定一个名称。

（2）各种约束：定义各种约束的语句，具体内容请参见 13.1 和 13.2 节介绍的各种约束的添加语法。

例 13.11 更新例 13.8 中的名称为 fk_classid 的外键约束为级联删除和级联更新，可以使用下面的代码。**（实例位置：光盘\TM\sl\13\13.11）**

```
ALTER TABLE tb_student1a DROP FOREIGN KEY fk_classid;
ALTER TABLE tb_student1a
ADD CONSTRAINT fk_classid FOREIGN KEY (classid)
REFERENCES tb_class(id)
ON DELETE CASCADE
ON UPDATE CASCADE
;
```

运行上述代码，其结果如图 13.11 所示。

```
mysql> use db_database13;
Database changed
mysql> create table tb_student1a (
    -> id int auto_increment,
    -> name varchar(30) not null,
    -> sex varchar(2),
    -> classid int not null,
    -> birthday date,
    -> remark varchar(100),
    -> CONSTRAINT pk_id PRIMARY KEY (id),
    -> CONSTRAINT fk_classid FOREIGN KEY (classid)
    -> REFERENCES tb_class(id)
    -> ON DELETE RESTRICT
    -> ON UPDATE RESTRICT
    -> );
Query OK, 0 rows affected (0.42 sec)

mysql> ALTER TABLE tb_student1a DROP FOREIGN KEY fk_classid;
Query OK, 0 rows affected (0.09 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE tb_student1a
    -> ADD CONSTRAINT fk_classid FOREIGN KEY (classid)
    -> REFERENCES tb_class(id)
    -> ON DELETE CASCADE
    -> ON UPDATE CASCADE
    -> ;
Query OK, 0 rows affected (0.58 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> _
```

图 13.11 更新外键约束

13.4 小 结

本章主要介绍了定义完整约束、命名完整性约束、删除完整性约束和修改完整性约束等内容。其中，定义完整性约束和命名完整性约束是本章的重点，需要读者认真学习、灵活掌握，这在以后的数据库设计中非常实用。

13.5　实践与练习

1．在创建用户信息表 tb_manager 时，将 id 字段设置为主键。**（答案位置：光盘\TM\sl\13\13.12）**

2．创建一个不添加任何外键的教师信息表 tb_teacher，然后通过 ALTER TABLE 语句为其添加一个名称为 fk_ departmentid 的外键约束。**（答案位置：光盘\TM\sl\13\13.13）**

第14章

存储过程与存储函数

（视频讲解：22 分钟）

存储过程和存储函数是在数据库中定义一些SQL语句的集合，然后直接调用这些存储过程和存储函数来执行已经定义好的SQL语句，可以避免开发人员重复编写相同的SQL语句。而且，存储过程和存储函数是在MySQL服务器中存储和执行的，可以减少客户端和服务器端的数据传输。本章将介绍存储过程和存储函数的含义、作用，以及创建、使用、查看、修改及删除存储过程和存储函数的方法。

通过阅读本章，读者可以：

- **掌握 MySQL 中存储过程和存储函数的创建方法**
- **掌握存储过程和函数的调用、查看、修改和删除的方法**

14.1　创建存储过程和存储函数

在数据库系统中，为了保证数据的完整性、一致性，同时也为提高其应用性能，大多数据库常采用存储过程和存储函数技术。MySQL 在 5.0 版本后，也应用了存储过程和存储函数。存储过程和存储函数经常是一组 SQL 语句的组合，这些语句被当作整体存入 MySQL 数据库服务器中。用户定义的存储函数不能用于修改全局库状态，但该函数可从查询中被唤醒调用，也可以像存储过程一样通过语句执行。随着 MySQL 技术的日趋完善，存储过程将和存储函数在以后的项目中得到广泛的应用。

14.1.1　创建存储过程

在 MySQL 中，创建存储过程的基本形式如下。

```
CREATE PROCEDURE sp_name ([proc_parameter[···]])
   [characteristic ···] routine_body
```

其中，sp_name 参数是存储过程的名称；proc_parameter 表示存储过程的参数列表；characteristic 参数指定存储过程的特性；routine_body 参数是 SQL 代码的内容，可以用 BEGIN···END 来标识 SQL 代码的开始和结束。

说明

proc_parameter 中的参数由 3 部分组成，它们分别是输入输出类型、参数名称和参数类型。其形式为[IN | OUT | INOUT]param_name type。其中，IN 表示输入参数；OUT 表示输出参数；INOUT 表示既可以输入也可以输出；param_name 参数是存储过程参数名称；type 参数指定存储过程的参数类型，该类型可以为 MySQL 数据库的任意数据类型。

一个存储过程包括名字、参数列表，还可以包括很多 SQL 语句集。下面创建一个存储过程，其代码如下。

```
delimiter //
create procedure proc_name (in parameter integer)
begin
declare variable varchar(20);
if parameter=1 then
set variable='MySQL';
else
set variable='PHP';
end if;
insert into tb (name) values (variable);
end;
```

MySQL 中存储过程的建立以关键字 create procedure 开始，后面紧跟存储过程的名称和参数。MySQL 的存储过程名称不区分大小写，如 PROCE1()和 proce1()代表同一存储过程名。存储过程名或存储函数名不能与 MySQL 数据库中的内建函数重名。

MySQL 存储过程的语句块以 begin 开始，以 end 结束。语句体中可以包含变量的声明、控制语句、SQL 查询语句等。由于存储过程内部语句要以分号结束，所以在定义存储过程前，应将语句结束标志“;”更改为其他字符，并且应降低该字符在存储过程中出现的机率，更改结束标志可以用关键字 delimiter 定义，例如：

```
mysql>delimiter //
```

存储过程创建之后，可用如下语句进行删除，参数 proc_name 指存储过程名。

```
drop procedure proc_name
```

下面创建一个名称为 count_of_student 的存储过程。首先，创建一个名称为 students 的 MySQL 数据库，然后创建一个名为 studentinfo 的数据表。数据表结构如表 14.1 所示。

表 14.1　studentinfo 数据表结构

字　段　名	类型（长度）	默　　认	额　　外	说　　明
sid	INT(11)		auto_increment	主键自增型 sid
name	VARCHAR(50)			学生姓名
age	VARCHAR(11)			学生年龄
sex	VARCHAR(2)	M		学生性别
tel	BIGINT(11)			联系电话

例 14.1　创建一个名称为 count_of_student 的存储过程，统计 studentinfo 数据表中的记录数。代码如下。（**实例位置：光盘\TM\sl\14\14.1**）

```
delimiter //
create procedure count_of_student(OUT count_num INT)
reads sql data
begin
select count(*) into count_num from studentinfo;
end
//
```

在上述代码中，定义一个输出变量 count_num，存储过程应用 SELECT 语句从 studentinfo 表中获取记录总数，最后将结果传递给变量 count_num。存储过程的执行结果如图 14.1 所示。

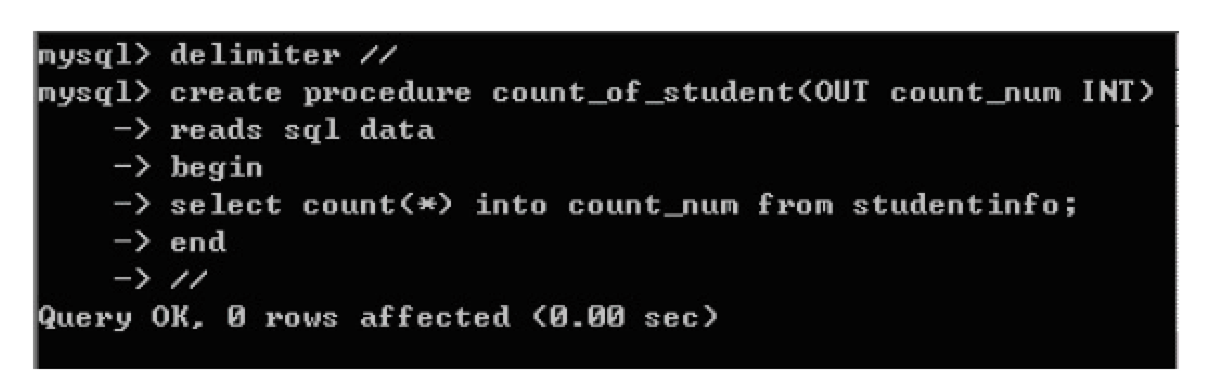


图 14.1　创建存储过程 count_of_student



代码执行完毕后，没有报出任何出错信息就表示存储函数已经创建成功。以后就可以调用这个存储过程，数据库中会执行存储过程中的 SQL 语句。

说明

MySQL 中默认的语句结束符为分号；存储过程中的 SQL 语句需要分号来结束。为了避免冲突，首先用 “DELIMITER //” 将 MySQL 的结束符设置为//，最后再用 “DELIMITER;” 来将结束符恢复成分号。这与创建触发器时是一样的。

14.1.2　创建存储函数

创建存储函数与创建存储过程大体相同，创建存储函数的基本形式如下。

```
CREATE FUNCTION sp_name ([func_parameter[,…]])
    RETURNS type
   [characteristic …] routine_body
```

创建存储函数的参数说明如表 14.2 所示。

表 14.2　创建存储函数的参数说明

参　　数	说　　明
sp_name	存储函数的名称
fun_parameter	存储函数的参数列表
RETURNS type	指定返回值的类型
characteristic	指定存储过程的特性
routine_body	SQL 代码的内容

func_parameter 可以由多个参数组成，其中每个参数均由参数名称和参数类型组成，其结构如下。

```
param_name type
```

param_name 参数是存储函数的函数名称；type 参数用于指定存储函数的参数类型。该类型可以是 MySQL 数据库所支持的类型。

例 14.2　同样，应用 studentinfo 表，创建名为 name_of_student 的存储函数，其代码如下。**（实例位置：光盘\TM\sl\14\14.2）**

```
delimiter //
create function name_of_student(std_id INT)
returns varchar(50)
begin
return(select name from studentinfo where sid=std_id);
end
//
```

上述代码中，存储函数的名称为 name_of_student；该函数的参数为 std_id；返回值是 VARCHAR 类型。该函数实现从 studentinfo 表查询与 std_id 相同 sid 值的记录，并将学生名称字段 name 中的值返回。存储函数的执行结果如图 14.2 所示。

```
mysql> delimiter //
mysql> create function name_of_student(std_id INT)
    -> returns varchar(50)
    -> begin
    -> return(select name from studentinfo where sid=std_id)
    -> end
    -> //
Query OK, 0 rows affected (0.03 sec)
```

图 14.2　创建 name_of_student()存储函数

14.1.3　变量的应用

MySQL 存储过程中的参数主要有局部参数和会话参数两种，这两种参数又可以被称为局部变量和会话变量。局部变量只在定义该局部变量的 begin…end 范围内有效，会话变量在整个存储过程范围内均有效。

1．局部变量

局部变量以关键字 DECLARE 声明，后跟变量名和变量类型，例如：

```
declare a int
```

当然，在声明局部变量时也可以用关键字 DEFAULT 为变量指定默认值，例如：

```
declare a int default 10
```

下述代码为读者展示如何在 MySQL 存储过程中定义局部变量以及其使用方法。在该例中，分别在内层和外层 BEGIN…END 块中都定义同名的变量 x，按照语句从上到下执行的顺序，如果变量 x 在整个程序中都有效，则最终结果应该都为 inner，但真正的输出结果却不同，这说明在内部 BEGIN…END 块中定义的变量只在该块内有效。

例 14.3　本例说明局部变量只在某个 BEGIN…END 块内有效，代码如下。**（实例位置：光盘\TM\sl\14\14.3）**

```
delimiter //
create procedure p1()
begin
declare x char(10) default 'outer ';
begin
declare x char(10) default 'inner ';
select x;
end;
select x;
end;
//
```

上述代码的运行结果如图 14.3 所示。

```
mysql> delimiter //
mysql> create procedure p1()
    -> begin
    -> declare x char(10) default 'outer ';
    -> begin
    -> declare x char(10) default 'inner ';
    -> select x;
    -> end;
    -> select x;
    -> end;
    -> //
Query OK, 0 rows affected (0.01 sec)
```

图 14.3　定义局部变量的运行结果

应用 MySQL 调用该存储过程的运行结果如图 14.4 所示。

```
mysql> call pl()//
+-------+
| x     |
+-------+
| inner |
+-------+
1 row in set (0.01 sec)

+-------+
| x     |
+-------+
| outer |
+-------+
1 row in set (0.03 sec)

Query OK, 0 rows affected (0.05 sec)
```

图 14.4　调用存储过程 pl()的运行结果

2．全局变量

MySQL 中的会话变量不必声明即可使用，会话变量在整个过程中有效，会话变量名以字符“@”作为起始字符。

例 14.4　分别在内部和外部 BEGIN…END 块中都定义了同名的会话变量@t，并且最终输出结果相同，从而说明会话变量的作用范围为整个程序。设置全局变量的代码如下。（**实例位置：光盘\TM\sl\14\14.4**）

```
delimiter //
create procedure p2()
begin
set @t=1;
begin
set @t=2;
select @t;
end;
select @t;
end;
//
```

上述代码的运行结果如图 14.5 所示。

```
mysql> delimiter //
mysql> create procedure p2()
    -> begin
    -> set @t=1;
    -> begin
    -> set @t=2;
    -> select @t;
    -> end;
    -> select @t;
    -> end;
    -> //
Query OK, 0 rows affected (0.00 sec)
```

图 14.5　设置全局变量

应用 MySQL 调用该存储过程的运行结果如图 14.6 所示。

```
mysql> call p2()//
+------+
| @t   |
+------+
| 2    |
+------+
1 row in set (0.01 sec)

+------+
| @t   |
+------+
| 2    |
+------+
1 row in set (0.01 sec)

Query OK, 0 rows affected (0.03 sec)
```

图 14.6　调用存储过程 p2()运行结果

3．为变量赋值

MySQL 中可以使用关键字 DECLARE 来定义变量，其基本语法如下。

```
DECLARE var_name[,···] type [DEFAULT value]
```

DECLARE 是用来声明变量的；var_name 参数是设置变量的名称。如果用户需要，也可以同时定义多个变量；type 参数用来指定变量的类型；DEFAULT value 的作用是指定变量的默认值，不对该参数进行设置时，其默认值为 NULL。

MySQL 中可以使用关键字 SET 为变量赋值，其基本语法如下。

```
SET var_name=expr[,var_name=expr] ···
```

SET 关键字是用来为变量赋值；var_name 参数是变量的名称；expr 参数是赋值表达式。一个 SET 语句可以同时为多个变量赋值，各个变量的赋值语句之间用“,”隔开。例如，为变量 mr_soft 赋值，代码如下。

```
SET mr_soft=10;
```

另外，MySQL 中还可以应用另一种方式为变量赋值，其语法结构如下。

```
SELECT col_name[,···] INTO var_name[,···] FROM table_name where condition
```



其中，col_name 参数标识查询的字段名称；var_name 参数是变量的名称；table_name 参数为指定数据表的名称；condition 参数为指定查询条件。例如，从 studentinfo 表中查询 name 为“LeonSK”的记录，并将该记录下的 tel 字段内容赋值给变量 customer_tel，其关键代码如下。

```
SELECT tel INTO customer_tel FROM studentinfo WHERE name= 'LeonSK ';
```



说明

上述赋值语句必须存在于创建的存储过程中，且需将赋值语句放置在 BEGIN…END 之间。若脱离此范围，该变量将不能使用或被赋值。

14.1.4　光标的运用

通过 MySQL 查询数据库，其结果可能为多条记录。在存储过程和函数中使用光标可以实现逐条读取结果集中的记录。光标使用包括声明光标（DECLARE CURSOR）、打开光标（OPEN CURSOR）、使用光标（FETCH CURSOR）和关闭光标（CLOSE CURSIR）。值得一提的是，光标必须声明在处理程序之前，且声明在变量和条件之后。

1．声明光标

在 MySQL 中，声明光标仍使用关键字 DECLARE，其语法如下。

```
DECLARE cursor_name CURSOR FOR select_statement
```

其中，cursor_name 是光标的名称，光标名称使用与表名同样的规则；select_statement 是一个 SELECT 语句，返回一行或多行数据。该语句也可以在存储过程中定义多个光标，但是必须保证每个光标名称的唯一性，即每一个光标必须有自己唯一的名称。

通过上述定义来声明光标 info_of_student，其代码如下。

```
DECLARE info_of_student CURSOR FOR SELECT
sid,name,age,sex,age
FROM studentinfo
WHERE sid=1;
```

说明

这里 SELECT 子句中不能包含 INTO 子句，并且光标只能在存储过程或存储函数中使用。上述代码并不能单独执行。

2．打开光标

在声明光标之后，要从光标中提取数据，必须首先打开光标。在 MySQL 中使用关键字 OPEN 来打开光标，其基本的语法如下。

```
OPEN cursor_name
```

其中，cursor_name 参数表示光标的名称。在程序中，一个光标可以打开多次。由于可能在用户打开光标后，其他用户或程序正在更新数据表，所以可能会导致用户在每次打开光标后，显示的结果都不同。

打开上面已经声明的光标 info_of_student，其代码如下。

```
OPEN info_of_student
```

3．使用光标

光标在顺利打开后，可以使用 FETCH…INTO 语句来读取数据，其语法如下。

```
FETCH   cursor_name INTO var_name[,var_name]…
```

其中，cursor_name 代表已经打开光标的名称；var_name 参数表示将光标中的 SELECT 语句查询出来的信息存入该参数中。var_name 是存放数据的变量名，必须在声明光标前定义好。FETCH…INTO 语句与 SELECT…INTO 语句具有相同的意义。

将已打开的光标 info_of_student 中 SELECT 语句查询出来的信息存入 tmp_name 和 tmp_tel 中。其中，tmp_name 和 tmp_tel 必须在使用前定义。其代码如下。

```
FETCH info_of_student INTO tmp_name,tmp_tel;
```

4．关闭光标

光标使用完毕后，要及时关闭，在 MySQL 中采用关键字 CLOSE 关闭光标，其语法格式如下。

```
CLOSE cursor_name
```

cursor_name 参数表示光标名称。下面关闭已打开的光标 info_of_student，其代码如下。

```
CLOSE info_of_student
```



说明

对于已关闭的光标，在其关闭之后则不能使用关键字 FETCH 来使用光标。光标在使用完毕后一定要关闭。

14.2 存储过程和存储函数的调用

存储过程和存储函数都是存储在服务器的 SQL 语句的集合。要使用这些已经定义好的存储过程和存储函数就必须要通过调用的方式来实现。对存储过程和函数的操作主要可以分为调用、查看、修改和删除。

14.2.1 调用存储过程

存储过程的调用在前面的示例中多次被用到。MySQL 中使用 CALL 语句来调用存储过程。调用存



储过程后，数据库系统将执行存储过程中的语句；然后将结果返回给输出值。CALL 语句的基本语法形式如下。

```
CALL sp_name([parameter[,…]]);
```

其中，sp_name 是存储过程的名称；parameter 是存储过程的参数。

14.2.2　调用存储函数

在 MySQL 中，存储函数的使用方法与 MySQL 内部函数的使用方法基本相同。用户自定义的存储函数与 MySQL 内部函数性质相同。区别在于，存储函数是用户自定义的，而内部函数由 MySQL 自带。其语法结构如下。

```
SELECT function_name([parameter[,…]]);
```

例 14.5 创建存储过程并在 PHP 中调用该存储过程实现用户注册。**（实例位置：光盘\TM\sl\14\14.5）**

（1）创建 pro_reg 存储过程，其代码如下。

```
delimiter //
create procedure pro_reg(in nc varchar(50),in pwd varchar(50),in email varchar(50),in address varchar(50))
begin
insert into tb_reg(name,pwd,email,address) values (nc,pwd,email,address);
end;
//
```

执行效果如图 14.7 所示。

```
mysql> delimiter //
mysql> create procedure pro_reg(in nc varchar(50),in pwd varchar(50),in email va
rchar(50),in address varchar(50))
    -> begin
    -> insert into tb_reg(name,pwd,email,address) values (nc,pwd,email,address);

    -> end;
    -> //
Query OK, 0 rows affected (0.02 sec)
```

图 14.7　创建存储过程

（2）通过 PHP 预定义类 mysqli 实现与 MySQL 数据库的连接，代码如下。

```
<?php
 if($_POST['submit']!=""){
    $conn=new mysqli("localhost","root","root","db_database14");          //连接数据库
    $conn->query("set names utf8");                                       //设置编码格式
    $name=$_POST['name'];
    $pwd=md5($_POST['pwd']);
    $email=$_POST['email'];
    $address=$_POST['address'];
```

（3）调用存储过程 pro_reg 实现将用户录入的注册信息保存到数据库，代码如下。

```
if($sql=$conn->query("call pro_reg('".$name."','".$pwd."','".$email."','".$address."')")){    //调用存储过程
        echo "<script>alert('用户注册成功!');</script>";
    }else{
        echo "<script>alert('用户注册失败!');</script>";
    }
}
?>
```

运行本实例，在文本框中输入如图 14.8 所示的注册信息后，单击“注册”按钮即可将用户填写的注册信息保存到数据库中，最终保存结果如图 14.9 所示。



图 14.8　录入注册信息

←T→			id	name	pwd	email	address
□	✎	✕	1	mrlzh	25d55ad283aa400af464c76d713c07ad	jlnu_lzh***@163.com	吉林长春

图 14.9　注册信息被存储到 MySQL 数据库

14.3　查看存储过程和存储函数

存储过程和存储函数创建以后，用户可以查看存储过程和存储函数的状态和定义。用户可以通过 SHOW STATUS 语句查看存储过程和存储函数状态，也可以通过 SHOW CREATE 语句来查看存储过程和存储函数的定义。

14.3.1　SHOW STATUS 语句

在 MySQL 中可以通过 SHOW STATUS 语句查看存储过程和存储函数的状态。其基本语法结构如下。

```
SHOW {PROCEDURE | FUNCTION}STATUS[LIKE 'pattern']
```



其中，PROCEDURE 参数表示查询存储过程；FUNCTION 参数表示查询存储函数；LIKE 'pattern' 参数用来匹配存储过程或存储函数名称。

14.3.2　SHOW CREATE 语句

MySQL 中可以通过 SHOW CREATE 语句来查看存储过程和函数的状态，其语法结果如下。

```
SHOW CREATE{PROCEDURE | FUNCTION } sp_name;
```

其中，PROCEDURE 参数表示存储过程；FUNCTION 参数表示查询存储函数；sp_name 参数表示存储过程或函数的名称。

例 14.6　下面查询名为 count_of_student 的存储过程，其代码如下。**（实例位置：光盘\TM\sl\14\14.6）**

```
show create procedure count_of_student ;
```

其运行结果如图 14.10 所示。

```
1 row in set (0.02 sec)

mysql> show create procedure  count_of_student//
+------------------+---------------------+-------------------------------------------
--------------------------------------------------------------------------------------
-----------------------------------------+
| Procedure        | sql_mode            | Create Procedure

                                         |
+------------------+---------------------+-------------------------------------------
--------------------------------------------------------------------------------------
-----------------------------------------+
| count_of_student | NO_AUTO_CREATE_USER | CREATE DEFINER=`root`@`localhost` PRO
CEDURE `count_of_student`(OUT count_num INT)
    READS SQL DATA
begin
select count(*) into count_num from studentinfo;
end |
```

图 14.10　应用 SHOW CREATE 语句查看存储过程

查询结果显示存储过程的定义、字符集等信息。

说明

SHOW STATUS 语句只能查看存储过程或函数所操作的数据库对象，如存储过程或函数的名称、类型、定义者、修改时间等信息，并不能查询存储过程或函数的具体定义。如果需要查看详细定义，需要使用 SHOW CREATE 语句。

14.4　修改存储过程和存储函数

修改存储过程和存储函数是指修改已经定义好的存储过程和函数。MySQL 中通过 ALTER

PROCEDURE 语句来修改存储过程，通过 ALTER FUNCTION 语句来修改存储函数。

MySQL 中修改存储过程和存储函数的语句的语法形式如下。

```
ALTER {PROCEDURE | FUNCTION} sp_name [characteristic ...]
characteristic:
  { CONTAINS SQL | NO SQL | READS SQL DATA | MODIFIES SQL DATA }
  | SQL SECURITY { DEFINER | INVOKER }
  | COMMENT 'string'
```

其参数说明如表 14.3 所示。

表 14.3　修改存储过程和存储函数的语法的参数说明

参　　数	说　　明
sp_name	存储过程或函数的名称
characteristic	指定存储函数的特性
CONTAINS SQL	表示子程序包含 SQL 语句，但不包含读写数据的语句
NO SQL	表示子程序不包含 SQL 语句
READS SQL DATA	表示子程序中包含读数据的语句
MODIFIES SQL DATA	表示子程序中包含写数据的语句
SQL SECURITY{DEFINER|INVOKER}	指明权限执行。DEFINER 表示只有定义者自己才能够执行；INVOKER 表示调用者可以执行
COMMENT 'string'	是注释信息

例 14.7　下面应用此语句修改存储过程 count_of_student，其代码如下。（**实例位置：光盘\TM\sl\14\14.7**）

```
alter procedure count_of_student
modifies sql data
sql security invoker;
```

其运行结果如图 14.11 所示。

```
mysql> alter procedure count_of_student
    -> modifies sql data
    -> sql security invoker;
    -> //
Query OK, 0 rows affected (0.00 sec)
```

图 14.11　修改存储过程 count_of_student 的定义



如果读者希望查看修改后的结果。可以应用 SELECT…FROM studentinfo.Ruotines WHERE ROUTINE_NAME='sp_name'来查看表的信息。由于篇幅限制，这里不进行详细讲解。



14.5　删除存储过程和存储函数

删除存储过程和存储函数指删除数据库中已经存在的存储过程或存储函数。MySQL 中通过 DROP PROCEDURE 语句来删除存储过程，通过 DROP FUNCTION 语句来删除存储函数。在删除之前，必须确认该存储过程或函数没有任何依赖关系，否则可能会导致其他与其关联的存储过程无法运行。

删除存储过程和存储函数的语法如下。

```
DROP {PROCEDURE | FUNCTION} [IF EXISTS] sp_name
```

其中，sp_name 参数表示存储过程或存储函数的名称；IF EXISTS 是 MySQL 的扩展，判断存储过程或函数是否存在，以免发生错误。

例 14.8　下面删除名称为 count_of_student 的存储过程，其关键代码如下。**（实例位置：光盘\TM\sl\14\14.8）**

```
drop procedure count_of_student;
```

删除存储过程 count_of_student 的运行结果如图 14.12 所示。

```
mysql> drop procedure count_of_student//
Query OK, 0 rows affected (0.02 sec)

mysql>
```

图 14.12　删除 count_of_student 存储过程

例 14.9　下面删除名称为 name_of_student 的存储函数，其关键代码如下。**（实例位置：光盘\TM\sl\14\14.9）**

```
drop function name_of_student;
```

删除存储函数 name_of_student 的运行结果如图 14.13 所示。

```
mysql> drop function name_of_student
    -> //
Query OK, 0 rows affected (0.00 sec)

mysql>
```

图 14.13　删除 name_of_student 存储函数

当返回结果没有提示警告或报错时，则说明存储过程或存储函数已经被顺利删除。用户可以通过查询 students 数据库下的 Routines 表来确认上面的删除是否成功。

14.6　小　　结

本章对 MySQL 数据库的存储过程和存储函数进行了详细讲解，存储过程和存储函数都是用户自

已定义的 SQL 语句的集合。它们都存储在服务器端，只要调用就可以在服务器端执行。本章重点讲解了创建存储过程和存储函数的方法。通过 CREATE PROCEDURE 语句来创建存储过程，通过 CREATE FUNCTION 语句来创建存储函数。这两个内容是本章的难点，需要读者将书中的知识点结合实际操作进行练习。

14.7 实践与练习

1. 将名称为 name_of_student 的存储函数的读写权限修改为 READS SQL DATA，并加上注释信息“FIND NAME”。**（答案位置：光盘\TM\sl\14\14.10）**

2. 通过 SHOW STATUS 语句查看存储函数的状态。**（答案位置：光盘\TM\sl\14\14.11）**

第15章

触　发　器

（视频讲解：22 分钟）

触发器是由事件来触发某个操作。这些事件包括 INSERT 语句、UPDATE 语句和 DELETE 语句。当数据库系统执行这些事件时，就会激活触发器执行相应的操作。本章将对触发器的含义、作用，以及创建、查看和删除触发器的方法进行详细介绍。

通过阅读本章，读者可以：

- **了解触发器的概念**
- **了解创建单条执行语句的触发器的方法**
- **掌握创建多条执行语句的触发器的方法**
- **掌握查看触发器的方法**
- **掌握使用触发器的方法**
- **掌握删除触发器的方法**

15.1　MySQL 触发器

触发器是由 MySQL 的基本命令事件来触发某种特定操作，这些基本的命令由 INSERT、UPDATE、DELETE 等事件来触发某些特定操作。满足触发器的触发条件时，数据库系统就会自动执行触发器中定义的程序语句，可以令某些操作之间的一致性得到协调。

15.1.1　创建 MySQL 触发器

在 MySQL 中，创建只有一条执行语句的触发器的基本形式如下。

```
CREATE   TRIGGER   触发器名  BEFORE | AFTER  触发事件
ON  表名  FOR EACH ROW  执行语句
```

具体的参数说明如下。

（1）触发器名指定要创建的触发器名字。

（2）参数 BEFORE 和 AFTER 指定触发器执行的时间。BEFORE 指在触发时间之前执行触发语句；AFTER 表示在触发时间之后执行触发语句。

（3）触发时间参数指数据库操作触发条件，其中包括 INSERT、UPDATE 和 DELETE。

（4）表名指定触发时间操作表的名称。

（5）FOR EACH ROW 表示任何一条记录上的操作满足触发事件都会触发该触发器。

（6）执行语句指触发器被触发后执行的程序。

例 15.1　下面创建一个由插入命令 INSERT 触发的触发器 auto_save_time，具体步骤如下。**（实例位置：光盘\TM\sl\15\15.1）**

（1）创建一个名称为 timelog 的表格，该表的结构非常简单。相关代码如下所示。

```
create table timelog(
id int(11) primary key auto_increment not null,
savetime varchar(50) not null
);
```

（2）创建名称为 auto_save_time 的触发器，其代码如下。

```
delimiter //
create trigger auto_save_time before insert
on studentinfo for each row
insert into timelog(savetime) values(now());
//
```

以上代码的运行结果如图 15.1 所示。



auto_save_time 触发器创建成功，其具体的功能是当用户向 studentinfo 表中执行 INSERT 操作时，数据库系统会自动在插入语句执行之前向 timelog 表中插入当前时间。下面通过向 studentinfo 表中插入一条信息来查看触发器的作用，其代码如下所示。

```
insert into studentinfo(name) values ('Chris');
```

然后执行 SELECT 语句查看 timelog 表中是否执行 INSERT 操作，其结果如图 15.2 所示。

```
mysql> delimiter //
mysql> create trigger auto_save_time before insert
    -> on studentinfo for each row
    -> insert into timelog(savetime) values(now());
    -> //
Query OK, 0 rows affected (0.00 sec)
```

图 15.1　创建 auto_save_time 触发器

```
mysql> select * from timelog //
+----+---------------------+
| id | savetime            |
+----+---------------------+
|  1 | 2011-06-07 15:44:55 |
+----+---------------------+
1 row in set (0.01 sec)
```

图 15.2　查看 timelog 表中是否执行插入操作

以上结果显示，在向 studentinfo 表中插入数据时，savetime 表中也会被插入一条当前系统时间的数据。

15.1.2　创建具有多条执行语句的触发器

15.1.1 节中已经介绍了如何创建一个最基本的触发器，但是在实际应用中，往往触发器中包含多条执行语句。其中创建具有多条执行语句的触发器语法结构如下。

```
CREATE TRIGGER 触发器名称 BEFORE | AFTER 触发事件
ON 表名 FOR EACH ROW
BEGIN
执行语句列表
END
```

其中，创建具有多条执行语句触发器的语法结构与创建触发器的一般语法结构大体相同，其参数说明请参考 15.1.1 节中的参数说明，这里不再赘述。在该结构中，将要执行的多条语句放入 BEGIN 与 END 之间。多条语句需要执行的内容，需要用分隔符“;”隔开。

说明

一般放在 BEGIN 与 END 之间的多条执行语句必须用结束分隔符“;”分开。在创建触发器过程中需要更改分隔符，故这里应用第 14 章提到的 DELIMITERT 语句，将结束符号变为“//”。当触发器创建完成后，读者同样可以应用该语句将结束符换回“;”。

例 15.2　创建一个由 DELETE 触发多条执行语句的触发器 delete_time_info。模拟一个删除日志数据表和一个删除时间表。当用户删除数据库中的某条记录后，数据库系统会自动向日志表中写入日志信息。创建具有多个执行语句的触发器的过程如下：

在例 15.1 中创建的 timelog 数据表基础上，另外创建一个名称为 timeinfo 的数据表，代码如下。（实

例位置：光盘\TM\sl\15\15.2）

```
create table timeinfo(
id int(11) primary key auto_increment,
info varchar(50) not null
)//
```

然后创建一个由 DELETE 触发多条执行语句的触发器 delete_time_info，其代码如下。

```
delimiter //
create trigger delete_time_info after delete
on studentinfo for each row
begin
insert into timelog(savetime) values (now());
insert into timeinfo(info) values ('deleteact');
end
//
```

运行以上代码的结果如图 15.3 所示。

```
mysql> delimiter //
mysql> create trigger delete_time_info after delete
    -> on studentinfo for each row
    -> begin
    -> insert into timelog(savetime) values (now());
    -> insert into timeinfo(info) values ('deleteact');
    -> end
    -> //
Query OK, 0 rows affected (0.05 sec)
```

图 15.3　创建具有多个执行语句的触发器 delete_time_info

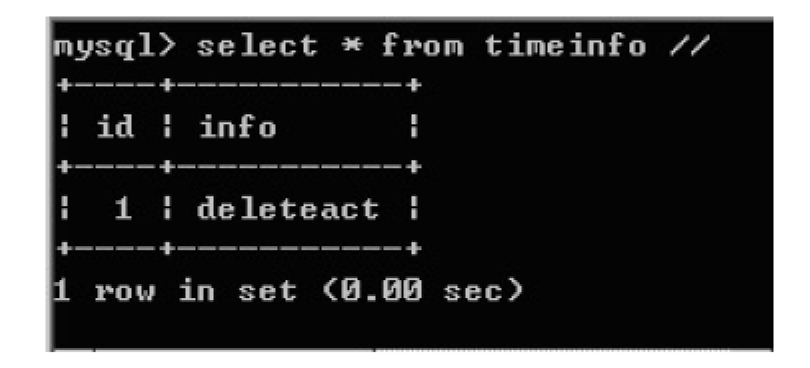
例 15.3　触发器创建成功，当执行删除操作后，timelog 与 timeinfo 表中将会插入两条相关记录。执行删除操作的代码如下。**（实例位置：光盘\TM\sl\15\15.3）**

```
DELETE FROM studentinfo where sid=7;
```

删除成功后，应用 SELECT 语句分别查看数据表 timelog 与数据表 timeinfo，其运行结果如图 15.4 和图 15.5 所示。

```
mysql> select * from timelog //
+----+---------------------+
| id | savetime            |
+----+---------------------+
|  2 | 2011-06-07 17:07:59 |
+----+---------------------+
1 row in set (0.00 sec)
```

图 15.4　查看数据表 timelog 信息



```
mysql> select * from timeinfo //
+----+-----------+
| id | info      |
+----+-----------+
|  1 | deleteact |
+----+-----------+
1 row in set (0.00 sec)
```

图 15.5　查看数据表 timeinfo 信息

从以上运行结果中可以看出，触发器创建成功后，当用户对 students 表执行 DELETE 操作时，students 数据库中的 timelog 数据表和 timeinfo 数据表中分别被插入操作时间和操作信息。





说明

在 MySQL 中，一个表在相同的时间和相同的触发时间只能创建一个触发器，如触发时间 INSERT，触发时间为 AFTER 的触发器只能有一个。但是可以定义 BEFORE 的触发器。

15.2　查看触发器

查看触发器是指查看数据库中已存在的触发器的定义、状态和语法等信息。查看触发器应用 SHOW TRIGGERS 语句。

15.2.1　SHOW TRIGGERS

在 MySQL 中，可以执行 SHOW TRIGGERS 语句查看触发器的基本信息，其基本形式如下。

```
SHOW TRIGGERS;
```

进入 MySQL 数据库，选择 students 数据库并查看该数据库中存在的触发器，其运行结果如图 15.6 所示。

```
mysql> show triggers;
+-------------------+---------+--------------+---------------------------------------------
------------------------------------------------------------------------------+---------+-
--------+---------------------------------------------+-------------------+
| Trigger           | Event   | Table        | Statement
                                                                               | Timing |
Created | sql_mode                                    | Definer           |
+-------------------+---------+--------------+---------------------------------------------
------------------------------------------------------------------------------+---------+-
--------+---------------------------------------------+-------------------+
| auto_save_time    | INSERT  | studentinfo  | insert into timelog(savetime) values
(now())                                                                        | BEFORE |
NULL    | NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION | root@localhost |
| delete_time_info  | DELETE  | studentinfo  | begin
insert into timelog(savetime) values (now());
insert into timeinfo(info) values ('deleteact');
end | AFTER  | NULL    | NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION | root@local
host |
+-------------------+---------+--------------+---------------------------------------------
------------------------------------------------------------------------------+---------+-
--------+---------------------------------------------+-------------------+
2 rows in set (0.16 sec)
```

图 15.6　查看触发器

在命令提示符中输入 SHOW TRIGGERS 语句即可查看选择数据库中的所有触发器，但是，应用该查看语句存在一定弊端，即只能查询所有触发器的内容，并不能指定查看某个触发器的信息。这样一

来，就会在用户查找指定触发器信息的时候带来极大不便。故推荐读者只在触发器数量较少的情况下应用 SHOW TRIGGERS 语句查询触发器基本信息。

15.2.2 查看 triggers 表中触发器信息

在 MySQL 中，所有触发器的定义都存在该数据库的 triggers 表中。读者可以通过查询 triggers 表来查看数据库中所有触发器的详细信息。查询语句如下所示。

```
SELECT * FROM information_schema.triggers;
```

其中，information_schema 是 MySQL 中默认存在的库，而 information_schema 是数据库中用于记录触发器信息的数据表。通过 SELECT 语句查看触发器信息。其运行结果与图 15.6 相同。但是如果用户想要查看某个指定触发器的内容，可以通过 WHERE 子句应用 TRIGGER 字段作为查询条件。其代码如下所示。

```
SELECT * FROM information_schema.triggers WHERE TRIGGER_NAME= '触发器名称';
```

其中，“触发器名称”这一参数为用户指定要查看的触发器名称，和其他 SELECT 查询语句相同，该名称内容需要用一对“''”（单引号）引用指定的文字内容。

说明

如果数据库中存在数量较多的触发器，建议读者使用第二种查看触发器的方式。这样会在查找指定触发器过程中避免很多麻烦。

15.3 使用触发器

在 MySQL 中，触发器按以下顺序执行：BEFORE 触发器、表操作、AFTER 触发器操作，其中表操作包括常用的数据库操作命令（如 INSERT、UPDATE、DELETE）。

例 15.4 触发器与表操作存在执行顺序，下面通过创建一个示例向读者展示三者的执行顺序关系。**（实例位置：光盘\TM\sl\15\15.4）**

（1）创建名称为 before_in 的 BEFORE INSERT 触发器，其代码如下。

```
create trigger before_in before insert on
studentinfo for each row
insert into timeinfo (info) values ('before');
```

（2）创建名称为 after_in 的 AFTER INSERT 触发器，其代码如下。

```
create trigger after_in after insert on
studentinfo for each row
insert into timeinfo (info) values ('after');
```



运行步骤（1）、（2）的结果如图 15.7 所示。

```
mysql> create trigger before_in before insert on
    -> studentinfo for each row
    -> insert into timeinfo (info) values ('before');
Query OK, 0 rows affected (0.02 sec)

mysql> create trigger after_in after insert on
    -> studentinfo for each row
    -> insert into timeinfo (info) values ('after');
Query OK, 0 rows affected (0.00 sec)
```

图 15.7 创建触发器运行结果

（3）创建完毕触发器，向数据表 studentinfo 中插入一条记录，其代码如下。

```
insert into studentinfo(name) values ('Nowitzki');
```

执行成功后，通过 SELECT 语句查看数据表 timeinfo 的插入情况，其代码如下。

```
select * from timeinfo;
```

运行以上代码，其运行结果如图 15.8 所示。

```
mysql> select * from timeinfo;
+----+--------+
| id | info   |
+----+--------+
|  2 | before |
|  3 | after  |
+----+--------+
2 rows in set (0.00 sec)
```

图 15.8 查看 timeinfo 表中触发器的执行顺序

查询结果显示 BEFORE 和 AFTER 触发器被激活。BEFORE 触发器首先被激活，然后 AFTER 触发器再被激活。



触发器中不能包含 START TRANSCATION、COMMIT 或 ROLLBACK 等关键字，也不能包含 CALL 语句。触发器执行非常严密，每一环都息息相关，任何错误都可能导致程序无法向下执行。已经更新过的数据表是不能回滚的，故在设计过程中一定要注意触发器的逻辑严密性。

15.4 删除触发器

在 MySQL 中，既然可以创建触发器，同样也可以通过命令删除触发器。删除触发器指删除原来已经在某个数据库中创建的触发器，与 MySQL 中删除数据库的命令相似，应用 DROP 关键字删除触

发器。其语法格式如下。

```
DROP TRIGGER 触发器名称
```

“触发器名称”参数为用户指定要删除的触发器名称，如果指定某个特定触发器名称，MySQL 在执行过程中将会在当前库中查找触发器。

说明

在应用完触发器后，切记一定要将触发器删除，否则在执行某些数据库操作时，会造成数据的变化。

例 15.5 下面将名称为 delete_time_info 的触发器删除，其执行代码如下。**（实例位置：光盘\TM\sl\15\15.5）**

```
DROP TRIGGER delete_time_info;
```

执行上述代码，其运行结果如图 15.9 所示。

```
mysql> use students;
Database changed
mysql> drop trigger delete_time_info;
Query OK, 0 rows affected (0.11 sec)
```

图 15.9　删除触发器

通过查看触发器命令来查看数据库 students 中的触发器信息，其代码如下。

```
SHOW TRIGGERS
```

查看触发器信息，可以从图 15.10 看出，名称为 delete_time_info 的触发器已经被删除。

```
mysql> show triggers;
+------------+--------+-------------+---------------------------------------------------------
---+--------+---------+---------------------------------------------------+-------------
----+
| Trigger    | Event  | Table       | Statement
   | Timing | Created | sql_mode                                          | Definer
   |
+------------+--------+-------------+---------------------------------------------------------
---+--------+---------+---------------------------------------------------+-------------
----+
| before_in  | INSERT | studentinfo | insert into timeinfo (info) values ('before
') | BEFORE | NULL    | NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION | root@localh
ost |
| after_in   | INSERT | studentinfo | insert into timeinfo (info) values ('after'
) | AFTER  | NULL    | NO_AUTO_CREATE_USER,NO_ENGINE_SUBSTITUTION | root@localh
ost |
+------------+--------+-------------+---------------------------------------------------------
---+--------+---------+---------------------------------------------------+-------------
----+
2 rows in set (0.11 sec)
```

图 15.10　查看 students 数据库中的触发器信息

注意

图 15.10 的返回结果显示，该数据库中存在两个触发器信息，这两个触发器是在 15.1.2 节中被创建的，如果用户在 db_database15 数据库中未创建该触发器，则返回结果会是一个“Empty set”。

15.5　小　　结

本章对 MySQL 数据库的触发器的定义和作用、创建触发器、查看触发器、使用触发器和删除触发器等内容进行了详细讲解，创建触发器和使用触发器是本章的重点内容。读者在创建触发器后，一定要查看触发器的结构。使用触发器时，触发器执行的顺序为 BEFORE 触发器、表操作（INSERT、UPDATE 和 DELETE）和 AFTER 触发器。读者需要将本章的知识结合实际需要来设计触发器。

15.6　实践与练习

1．创建一个由 INSERT 触发的触发器，实现当向 department 表中插入数据时，自动向 tb_students 表中插入当前时间。**（答案位置：光盘\TM\sl\15\15.6）**

2．删除原有的触发器，触发器删除完成后，执行 SELECT 语句来查看触发器是否还存在。**（答案位置：光盘\TM\sl\15\15.7）**

第16章

事务的应用

（视频讲解：15 分钟）

在操作 MySQL 过程中，对于一般简单的业务逻辑或中小型程序而言，无须考虑应用 MySQL 事务。但在比较复杂的情况下，往往在执行某些数据操作过程中，需要通过一组 SQL 语句执行多项并行业务逻辑或程序，这样，就必须保证所用命令执行的同步性，使执行序列中产生依靠关系的动作能够同时操作成功或同时返回初始状态。在此情况下，就需要优先考虑使用 MySQL 事务处理。

通过阅读本章，读者可以：

- 了解 MySQL 事务的概述
- 了解 MySQL 事务的创建与存在周期
- 掌握 MySQL 事务的查询和提交（回滚）
- 掌握 MySQL 事务行为
- 掌握 MySQL 事务的性能
- 掌握应用 MySQL 伪事务的方法

16.1　MySQL 事务概述

在 MySQL 中，事务由单独单元的一条或多条 SQL 语句组成。在这个单元中，每条 MySQL 语句是相互依赖的。而整个单独单元作为一个不可分割的整体，如果单元中一旦某条 SQL 语句执行失败或产生错误，整个单元将会回滚，所有受到影响的数据将返回到事务开始以前的状态；如果单元中的所有 SQL 语句均执行成功，则事务被顺利执行。

在现实生活中，事务处理数据的应用非常广泛，如网上交易、银行事务等。下面通过网上交易流程向读者展示事务的概念。

相信大多数用户都有过网上购物的体验，即用户登录某个大型购物网站，浏览该网站中所陈列的商品信息，将喜欢的商品放入购物车中，选购完毕后，用户需要对选购的商品进行在线支付，当用户对所选商品付款完毕，通知商家发货，在此过程中。用户所付货款并未提交到商户手中，当用户收到货物之后，可以确认收货，商家才收到商品货款，整个交易过程才算完成。如果任何一步操作失败，则都会导致双方陷入尴尬的境界，试想当用户选购商品，付款操作完成后，用户选择在发货过程中取消订单。这时商家并没有得到货款将取消操作，如果不应用事务处理，则用户在取消订单操作过程后，商家仍然继续将用户所订购的商品发送给用户，这会导致一些不愉快的争端。故在整个交易过程中，必须采用事务来对网上交易进行回滚操作。其流程如图 16.1 所示。

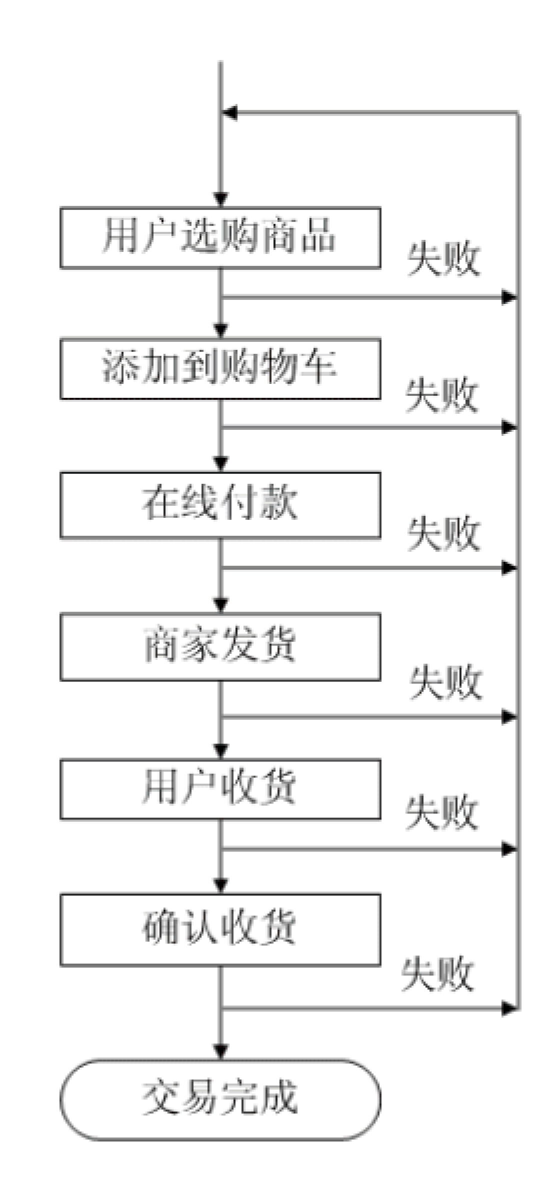


图 16.1　应用事务处理网上交易流程

在网上交易流程过程中，商家与用户的交易可以被认为是一个事务处理过程，如果在交易流程中存在一个环节失败，都可能导致双方交易的失败。如前面事务定义所说，所有这些流程都应该被成功执行，在 MySQL 中如果有任何命令失败，都会导致所有操作命令被撤销。系统返回未操作前的状态，即回滚到初始状态。添加到购物车、在线付款、商家发货等构成了一个基本的事务。整个交易流程可以被看作一个完整的单元，用于实现整体事务。

通过 InnoDB 和 BDB 类型表，MySQL 事务能够完全满足事务安全的 ACID 测试。但是并不是所有表类型都支持事务，如 MyISAM 类型表就不能支持事务，只能通过伪事务对表实现事务处理。



ACID 指出每个事务型 RDBMS 必须遵守的 4 个属性，即原子性、一致性、孤立性和持久性。

16.1.1 原子性

这就类似与化学中的原子，事务也具备整体性和不可分割性，被认为是一个不可分割的单元。假设一个事务由多种任务组成，其中的语句必须同时操作成功，才可以认为事务是成功的，否则将回滚到初始状态。从上面网上交易的例子可以看出，所有操作的成功是保证交易完成的前提条件，当任何一个环节出现问题时，就导致事务回滚，不能正常完成交易过程。即所有事务共同进退，保证了事务的整体性，这就是事务的原子性。

原子的执行是一个全部发生或全部失败的整体过程。在一个原子操作中，如果事务中的任何一条语句失败，前面执行的语句都将被返回，以保证数据的整体性不被破坏。这在常用的系统应用中，为保证数据的安全性起到一定作用。

16.1.2 一致性

在 MySQL 事务处理过程中，无论事务是完全成功或是在中途因某些环节失败而导致失败，但事务使系统处于一致的状态时，其必须保证一致性。如在网上进行转账操作的过程中，用户 A 向用户 B 的账户中转入 5 000 元，但用户 B 在查询转账信息的时候，发现自己的账户中只增加了 3 000 元，这样不能使整个事务达到一致性。例如，从上述网上交易的例子考虑，当交易完成后，商家的货物会减少，不可能出现当商家给用户发货后货物数量不变，而用户收到货物不增加的情况。

在 MySQL 中，一致性主要由 MySQL 的日志机制处理，它记录数据库的所有变化，为事务回复提供跟踪记录。如果系统在事务处理中间发生错误，MySQL 恢复过程将使用这些日志发现事务是否已经完全成功执行或需要返回。一致性属性保证数据库从不返回一个未处理的事务。

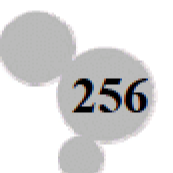
16.1.3 孤立性

孤立性是指每个事务在自己的空间发生，与其他发生在系统中的事务隔离，而且事务的结果只在它完全被执行时才能看到。即使这样的一个系统中同时发生多个事务，孤立性也可以保证特定的事务在完成之前，其结果是不被公布的。

当系统支持多个同时存在的用户和连接时，系统必须遵守孤立性原则，否则在执行过程中可能导致大量数据被破坏，孤立性保证每个事务完整地在其各自的空间内被顺利执行，保证事务与事务之间不会相互冲突。

16.1.4 持久性

在 MySQL 中，即便是数据库系统崩溃，一个被提交的事务仍然在坚持。当一个事务完成，数据



库的日志已经被更新时，持久性即可发挥其特有功效。在 MySQL 中，如果系统崩溃或者数据存储介质被破坏，通过使用日志，系统能够恢复在重启前进行的最后一次成功更新，可以反映系统崩溃时处于执行过程的事务的变化。

MySQL 的持久性是通过一条记录事务过程中系统变化的二进制事务日志文件来实现的。如果遇到硬件损坏或者系统的异常关机，系统在下一次启动时，通过使用最后的备份和日志就可以恢复丢失数据。



说明

默认情况下，InnoDB 表持久性最久，MyISAM 表提供部分持久。

16.2　MySQL 事务的创建与存在周期

通过上述对事务定义的叙述和事务特性的讲解，相信读者已经对事务有一个初步的认识。下面的内容将对事务的存在周期做详细讲解。首先，向读者展示在 MySQL 中如何创建事务。

创建事务的一般过程是：初始化事务、创建事务、应用 SELECT 语句查询数据是否被录入和提交事务。如果用户不在操作数据库完成后执行事务提交，则系统会默认执行回滚操作。如果用户在提交事务前选择撤销事务，则用户在撤销前的所有事务将被取消，数据库系统会回到初始状态。

注意

默认情况下，在 MySQL 中创建的数据表类型都是 MyISAM，但是该类型的数据表并不能支持事务。所以，如果用户想让数据表支持事务处理能力，必须将当前操作数据表的类型设置为 InnoDB 或 BDB。

在创建事务的过程中，用户需要创建一个 InnoDB 或 BDB 类型的数据表，其基本命令结构如下。

```
CREATE TABLE   table_name(field_defintions) TYPE = INNODB/BDB;
```

其中，table_name 为表名，而 field_defintions 为表内定义的字段等属性，TYPE 指定数据表的类型，既可以是 InnoDB 类型，也可以是 BDB 类型。

当用户希望将已经存在的表支持事务处理，则用户可以应用 ALTER TABLE 命令，指定数据表的类型即可实现对表的类型更改操作，使原本不支持事务的数据表更改为支持事务处理的类型，其命令如下。

```
ALTER TABLE   table_name TYPE= INNODB/BDB;
```

在用户更改完表的类型后，即可使数据表支持事务处理。

说明

应用 ALTER TABLE 操作可能会导致数据库中的数据丢失，因此为了避免非预期结果出现，在使用 ALTER TABLE 命令之前，用户需要创建一个表备份。

16.2.1　初始化事务

初始化 MySQL 事务，首先声明初始化 MySQL 事务后所有的 SQL 语句为一个单元。在 MySQL 中，应用 START TRANSACTION 命令来标记一个事务的开始，初始化事务的结构如下。

```
START TRANSACTION;
```

另外，用户也可以使用 BEGIN 或者 BEGIN WORK 命令初始化事务，通常 START TRANSACTION 命令后面跟随的是组成事务的 SQL 语句。

在命令提示符中输入如下命令。

```
start transaction;
```

如果在用户输入以上代码后，MySQL 数据库没有给出警告提示或返回错误信息，则说明用户已经事务初始化成功，用户可以继续执行下一步操作。

16.2.2　创建事务

例 16.1　初始化事务成功后，可以创建事务。这里以向名称为 connection 的数据表中插入一条记录为例，讲解事务的创建。首先打开数据库，选定某个数据库，然后初始化事务，最后创建事务，向指定的数据表中添加记录，其代码如下。**（实例位置：光盘\TM\sl\16\16.1）**

```
mysql –uroot –proot
use db_database16 ;
start transaction;
insert into connection(email,cellphone,QQ,sid)
values('barrystephen@126.com',13456000000,187034000,3);
```

其运行结果如图 16.2 所示。

```
mysql> start transaction;
Query OK, 0 rows affected (0.00 sec)

mysql> insert into connection(email,cellphone,QQ,sid)
    -> values('barrystephen@126.com',13456000000,187034000,3);
Query OK, 1 row affected (0.02 sec)
```

图 16.2　创建事务

16.2.3　应用 SELECT 语句查询数据是否被正确录入

事务创建成功后，建议读者通过 SELECT 语句查询数据是否被正确录入。

例 16.2　在事务初始化成功后，用户创建事务，继续在命令提示符中输入如下指令。**（实例位置：**



光盘\TM\sl\16\16.2）

```
SELECT * FROM connection WHERE   sid=3;
```

其运行结果如图 16.3 所示。

```
mysql> SELECT * FROM connection WHERE  sid=3;
+----+----------------------+-------------+-----------+-----+
| id | email                | cellphone   | QQ        | sid |
+----+----------------------+-------------+-----------+-----+
|  5 | barrystephen@126.com | 13456000000 | 187034000 |   3 |
+----+----------------------+-------------+-----------+-----+
1 row in set (0.00 sec)
```

图 16.3　查询数据是否被正确录入



在用户插入新表为 InnoDB 类型或更改原来表类型为 InnoDB 时，如果在输入命令提示后，MySQL 提示“The 'InnoDB' feature is disabled; you need 'InnoDB' to have it working”警告，则说明 InnoDB 表类型并没有被开启，用户需要找到 MySQL 文件目录下的 my.ini 文件，定位 skip_innodb 选项位置，将原来的 skip_innodb 改为“#skip_innodb”后保存该文件，重新启动 MySQL 服务器，即可令数据库支持 InnoDB 类型表。

16.2.4　提交事务

在用户没有提交事务之前，当其他用户连接 MySQL 服务器时，应用 SELECT 语句查询结果，则不会显示没有提交的事务。当且仅当用户成功提交事务后，其他用户才可能通过 SELECT 语句查询事务结果。由事务的特性可知，事务具有孤立性，当事务处在处理过程中，其实 MySQL 并未将结果写入磁盘中，这样一来，这些正在处理的事务相对其他用户是不可见的。一旦数据被正确插入，用户可以使用 COMMIT 命令提交事务。提交事务的命令结构如下。

```
COMMIT
```

一旦当前执行事务的用户提交当前事务，则其他用户就可以通过会话查询结果。

16.2.5　撤销事务（事务回滚）

撤销事务，又被称作事务回滚，即事务被用户开启、用户输入的 SQL 语句被执行后，如果用户想要撤销刚才的数据库操作，可使用 ROLLBACK 命令撤销数据库中的所有变化。ROLLBACK 命令结构如下。

```
ROLLBACK
```

输入回滚操作后，如何判断是否执行回滚操作了呢？可以通过 SELECT 语句查看 16.2.2 节中插入的数据是否存在，其运行结果如图 16.4 所示。

```
mysql> select * from connection where sid=3;
Empty set (0.00 sec)

mysql>
```

图 16.4　执行回滚操作后应用 SELECT 查询

注意

如果执行一个回滚操作，则在输入 START TRANSACTIONA 命令后的所有 SQL 语句都将执行回滚操作。故在执行事务回滚前，用户需要慎重选择执行回滚操作。如果用户开启事务后，没有提交事务，则事务默认为自动回滚状态，即不保存用户之前的任何操作。

说明

在现实应用中，事务撤销即事务回滚有着很重要的意义。例如，用户 A 和用户 B 采用银行转账方式交易，用户 A 将个人账户的部分存款转移到用户 B 的个人账户过程中，若银行的数据库系统突然发生错误或异常，则交易事务提交失败，系统执行回滚操作，恢复到交易的初始状态。这样，采用事务回滚可以避免因特殊情况而导致事务提交失败，从而导致不必要的损失。

16.2.6　事务的存在周期

事务的周期由用户在命令提示符中输入 START TRANSACTION 指令开始，直至用户输入 COMMIT 结束，图 16.5 展示了一个简单事务存在周期流程图。

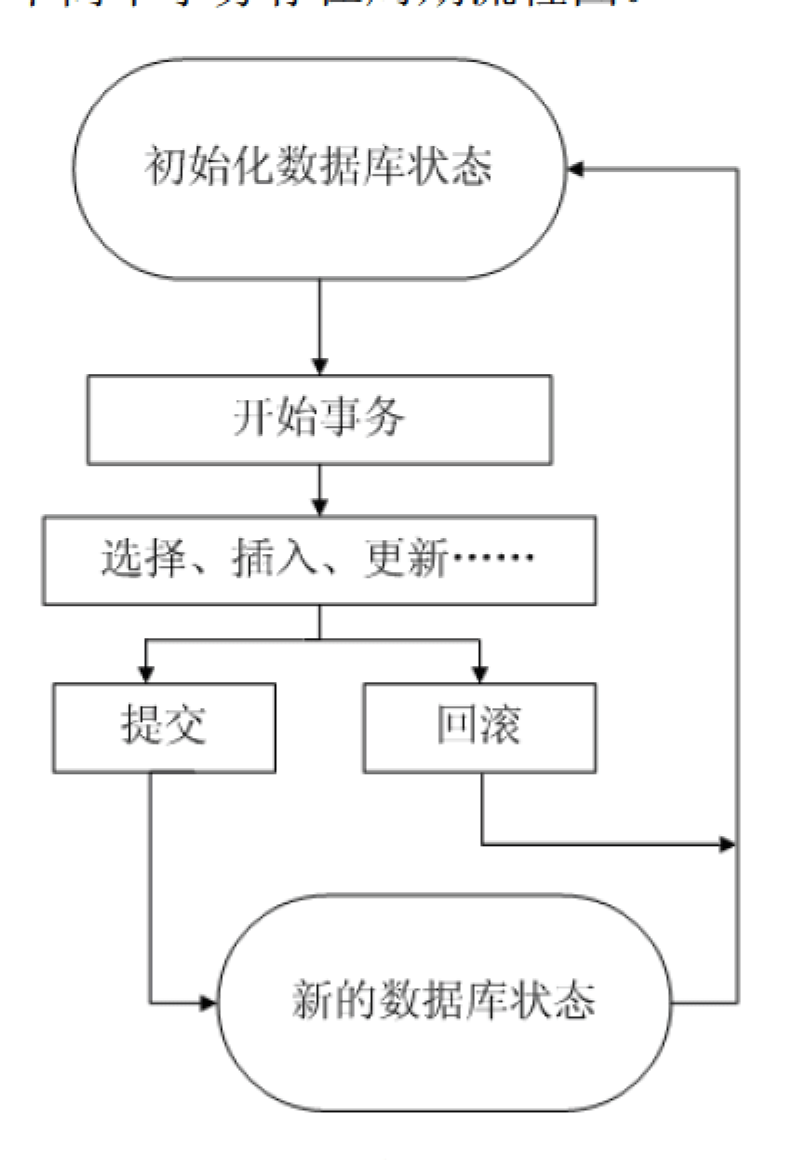


图 16.5　事务的存在周期



事务不支持嵌套功能，当用户在未结束第一个事务又重新打开一个事务时，则前一个事务会自动提交。在 MySQL 中很多命令都会隐藏执行 COMMIT 命令。

16.3　MySQL 事务行为

在 MySQL 中，存在两个可以控制行为的变量，它们分别是 AUTOCOMMIT 变量和 TRANSACTION ISOLACTION LEVEL 变量。

16.3.1　自动提交

在 MySQL 中，如果不更改其自动提交变量，则系统会自动向数据库提交结果，用户在执行数据库操作过程中，不需要使用 START TRANSACTION 语句开始事务，应用 COMMIT 或者 ROLLBACK 提交事务或执行回滚操作。如果用户希望通过控制 MySQL 自动提交参数，可以更改提交模式，这一更改过程是通过设置 AUTOCOMMIT 变量来实现的。

下面通过一个示例向读者展示如何关闭自动提交参数，在命令提示符中输入以下命令。

```
SET AUTOCOMMIT=0 ;
```

关闭自动提交功能。只有当用户输入 COMMIT 命令后，MySQL 才将数据表中的资料提交到数据库中，如果不提交事务，而终止 MySQL 会话，数据库将会自动执行回滚操作。

例 16.3　在关闭自动提交命令后，查询数据表中数据，并且向数据表中添加一条记录，其命令如下。**（实例位置：光盘\TM\sl\16\16.3）**

```
select * from timeinfo;
insert into timeinfo(info) values('test autocommit');
```

以上代码的运行结果如图 16.6 所示。

```
mysql> set autocommit=0;
Query OK, 0 rows affected (0.00 sec)

mysql> select * from timeinfo;
Empty set (0.00 sec)

mysql> insert into timeinfo(info) values('test autocommit');
Query OK, 1 row affected (0.00 sec)

mysql> exit
Bye
```

图 16.6　取消自动提交操作

由于用户已经关闭自动提交功能，所以图 16.6 中所示的添加操作由于没有执行事务的提交操作，导致数据没有成功添加。再次查询数据表中的数据，运行结果如图 16.7 所示。

结果表明，之前插入的数据并未插入到数据库中，另外，可以通过查看@@AUTOCOMMIT 变量来查看当前自动提交状态，查看此变量同样应用 SELECT 语句，其运行结果如图 16.8 所示。

```
mysql> select * from timeinfo;
Empty set (0.00 sec)
```

图 16.7 应用 SELECT 语句查询自动提交关闭后的数据



```
mysql> select @@autocommit;
+--------------+
| @@autocommit |
+--------------+
|            0 |
+--------------+
1 row in set (0.00 sec)
```

图 16.8 查看自动提交变量

16.3.2 事务的孤立级

事务具有独立的空间，在 MySQL 服务器中，用户通过不同的会话执行不同的事务，在多用户环境中，许多 RDBMS 会话在任意指定时刻都是活动的。为了使这些事务互不影响，保证数据库性能不受到影响，采用事务的孤立级是十分有必要的。

孤立级在整个事务中起到了很重要的作用，如果没有这些事务的孤立性，不同的 SELECT 语句将会在同一事务的环境中检索到不同的结果，这将导致数据的不一致性。给不同的用户造成困扰，这样一来，用户就不能将查询的结果集作为计算基础。所以孤立性强制保持每个事务的独立性，以此来保证事务中看到一致的数据。

基于 ANSI/ISO SQL 规范，MySQL 提供以下 4 种孤立级。

（1）SERIALIZABLE（序列化）：顾名思义，以序列的形式对事务进行处理，该孤立级的特点是只有当事务提交后，用户才能从数据库中查看数据的变化。该孤立级运行会影响 MySQL 的性能。因为需要占用大量资源，以保证使大量事务在任意时间不被用户看到。

（2）REPEATABLE READ（可重读）：对于应用程序的安全性作出部分妥协，以提高其性能。事务在该孤立级上不会被看成一个序列，不过当前在执行事务的过程中，用户仍然看不到事务的过程。直到事务提交为止，用户才能够看到事务的变化结果。

（3）READ COMMITTED（提交后读）：提交后读孤立级的安全性比重复读安全性要低。在这一级的事务，用户可以看到其他事务添加的新记录。在事务处理时，如果存在其他用户同时对事务的相应表进行修改，那么在同一事务中不同时间内，应用 SELECT 语句可能返回不同的结果集。

（4）READ UNCOMMITTED（未提交读）：该孤立级提供事务之间的最小程度间隔，该孤立级容易产生虚幻读操作。其他用户可以在该孤立级上看到未提交的事务。

16.3.3 修改事务的孤立级

在 MySQL 中，可以使用 TRANSACTION ISOLATION LEVEL 变量修改事务孤立级，其中，MySQL



的默认孤立级为 REPEATABLE READ（可重读），用户可以使用 SELECT 命令获取当前事务孤立级变量的值，其命令如下。

```
SELECT @@tx_isolation ;
```

例 16.4　如果用户希望修改事务的孤立级，可以通过 SET 命令设置其不同值来修改事务的孤立级，操作命令如图 16.9 所示。**（实例位置：光盘\TM\sl\16\16.4）**

```
mysql> set global transaction isolation level
    -> read committed;
Query OK, 0 rows affected (0.00 sec)
```

图 16.9　设置事务孤立级



如果用户想修改事务的孤立级，必须首先获取 SUPER 优先权，以便用户可以顺利执行修改操作。

16.4　事务的性能

从 16.3 节中可以看出，应用不同孤立级的事务可能会对系统造成一系列影响，采用不同孤立级处理事务，可能会对系统稳定性和安全性等诸多因素造成影响。另外，有些数据库操作中，不需要应用事务处理，则用户在选择数据表类型时，需要选择合适的数据表类型。所以，在选择表类型时，应该考虑数据表具有完善的功能，且高效执行的前提下，也不会对系统增加额外的负担。

16.4.1　应用小事务

应用小事务的意义在于：保证每个事务不会在执行前等待很长时间，从而避免各个事务因为互相等待而导致系统性能的大幅度下降。用户在应用少数大事务的时候，可能无法看出因事务间互相等待而导致系统性能下降，但是当系统中存在处理量很大的数据库或多种复杂事务的时候，用户就可以明显感觉到事务因长时间等待，而导致系统性能下降。所以，应用小事务可以保证系统的性能，其可以快速变化或退出，这样，其他在队列中准备就绪的事务就不会受到明显影响。

16.4.2　选择合适的孤立级

因为事务的性能与其对服务器产生的负载成反比，即当事务孤立级越高，其性能越低，但是其安全性也越高。其性能和孤立级关系如图 16.10 所示。所以只有选择适当的孤立级，才能有效地提高 MySQL 系统性能和应用性。

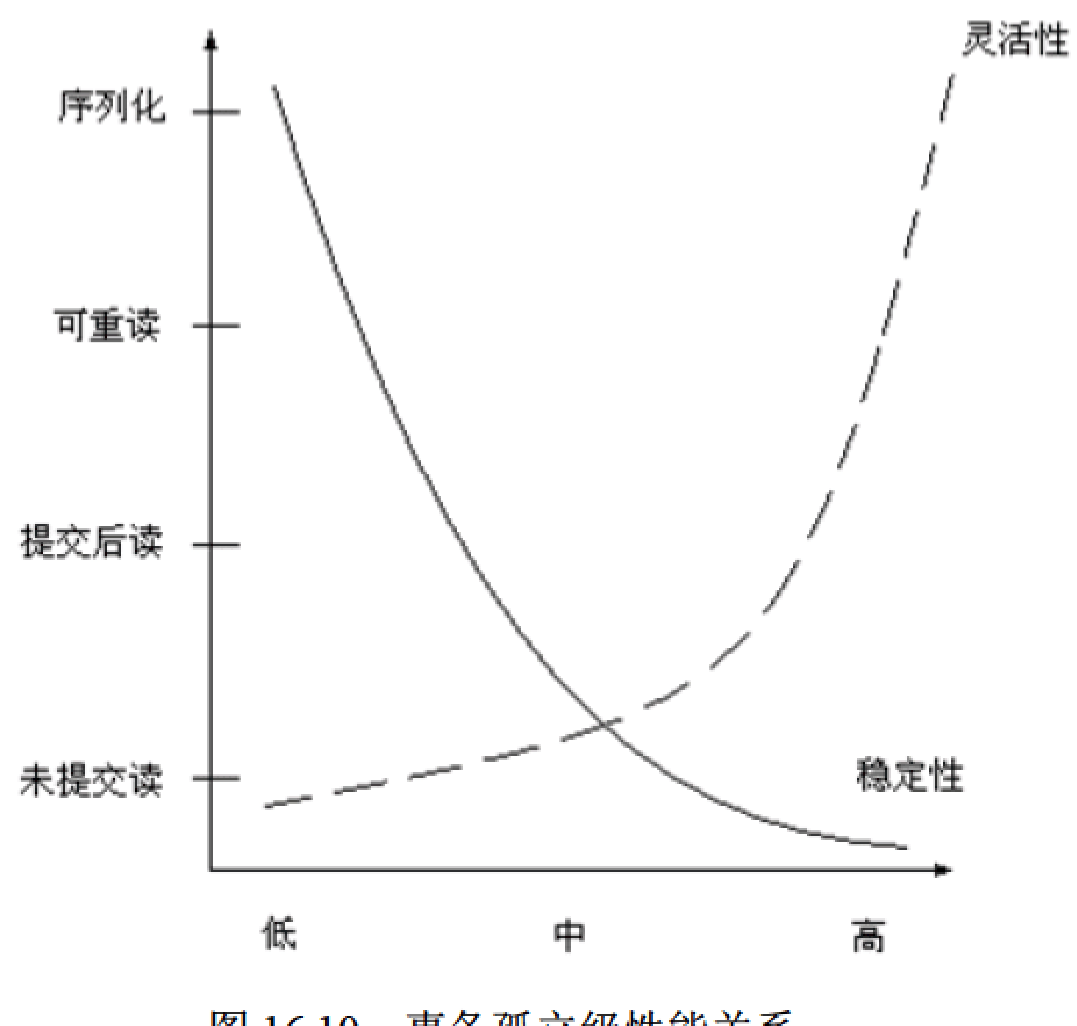


图 16.10　事务孤立级性能关系

从图 16.10 可以看出，虽然随着孤立级的增高稳定性也会随之改变，但是并不代表孤立级的稳定性越高，也不能表明其灵活性越高，故用户在选择孤立级的时候，需要根据自身实际情况选择适合应用的孤立级，切勿生搬硬套。

16.4.3　死锁的概念与避免

死锁，即当两个或者多个处于不同序列的用户打算同时更新某相同的数据库时，因互相等待对方释放权限而导致双方一直处于等待状态。在实际应用中，两个不同序列的客户打算同时对数据执行操作，极有可能产生死锁。更具体地讲，当两个事务相互等待操作对方释放所持有的资源，而导致两个事务都无法操作对方持有的资源，这样无限期的等待被称作死锁。

不过，MySQL 的 InnoDB 表处理程序具有检查死锁这一功能，如果该处理程序发现用户在操作过程中产生死锁，该处理程序立刻通过撤销操作来撤销其中一个事务，以便使死锁消失。这样就可以使另一个事务获取对方所占有的资源而执行逻辑操作。

16.5　MySQL 伪事务

在 MySQL 中，InnoDB 和 BDB 类型表可以支持事务处理，但是 MySQL 中 MyISAM 类型表并不能支持事务处理，对于某些应用该类型的数据表，用户可以选择应用表锁定来替代事务。这种引用表锁定来替代事务的事件被称作伪事务。使用表锁来锁定表的操作，可以加强非事务表在执行过程中的安全性和稳定性。

16.5.1　用表锁定代替事务

在 MySQL 的 MyISAM 类型数据表中，并不支持 COMMIT（提交）和 ROLLBACK（回滚）命令。当用户对数据库执行插入、删除、更新等操作时，这些变化的数据都被立刻保存在磁盘中。这样，在多用户环境中，会导致诸多问题，为了避免同一时间有多个用户对数据库中指定表进行操作。可以应用表锁定来避免在用户操作数据表过程中受到干扰。当且仅当该用户释放表的操作锁定后，其他用户才可以访问这些修改后的数据表。

设置表锁定代替事务基本步骤如下。

（1）为指定数据表添加锁定，其语法如下。

```
LOCK TABLES table_name lock_type,…
```

其中，table_name 为被锁定的表名，lock_type 为锁定类型，该类型包括以读方式（READ）锁定表；以写方式（WRITE）锁定表。

（2）用户执行数据表的操作，可以添加、删除或者更改部分数据。

（3）用户完成对锁定数据表的操作后，需要对该表进行解锁操作，释放该表的锁定状态，其语法如下。

```
UNLOCK TABLES
```

下面通过实例向读者展示如何以读方式锁定数据表和以写方式锁定数据表。

1. 以读方式锁定数据表

以读方式锁定数据表，该方式是设置锁定用户的其他方式操作，如删除、插入、更新都不被允许，直至用户进行解锁操作。

例 16.5　演示以读方式锁定表后，向该表中插入数据，以及解锁后再插入数据的情况。**（实例位置：光盘\TM\sl\16\16.5）**

首先，以锁定数据表 studentinfo 为例，在命令提示符下输入如下代码。

```
lock table studentinfo read;
```

然后，应用 SELECT 语句查看表中的信息，其运行结果如图 16.11 所示。

```
mysql> select * from studentinfo;
+-----+--------------+-----+-----+-----------+------+
| sid | name         | age | sex | tel       | time |
+-----+--------------+-----+-----+-----------+------+
|   1 | LeonSK       | 21  | M   | 123456789 |      |
|   2 | HarryMason   | 22  | M   | 135791234 |      |
|   3 | BarryStephen | 22  | M   |  95135746 |      |
|   4 | JillJason    | 20  | F   |  75147631 |      |
|   5 | MrSoft       | 12  | M   |  12345678 |      |
|   6 | Chris        | 29  | M   | 654321789 |      |
|   8 | Nowitzki     | 35  | M   |  15487658 |      |
+-----+--------------+-----+-----+-----------+------+
7 rows in set (0.00 sec)
```

图 16.11　查看以读方式锁定的 studentinfo 表

下面尝试向该数据表中插入一条数据，其运行结果如图 16.12 所示。

```
mysql> insert into studentinfo (name,age,sex,tel)values
    -> ('LarryBird','29','M',1874930203);
ERROR 1100 (HY000): Table 'timeinfo' was not locked with LOCK TABLES
mysql>
```

图 16.12　向以读方式锁定的表中插入数据

从上述结果可以看出，当用户试图向数据库插入数据时，将会返回失败信息。当用户将锁定的表解锁后，再次执行插入操作，其运行结果如图 16.13 所示。

```
mysql> unlock tables;
Query OK, 0 rows affected (0.00 sec)

mysql> insert into studentinfo (name,age,sex,tel)values
    -> ('LarryBird','29','M',1874930203);
Query OK, 1 row affected, 1 warning (0.08 sec)
```

图 16.13　向解锁后的数据表中添加数据

锁定被释放后，用户可以对数据库执行添加、删除、更新等操作。

说明

其中，lock_type 参数中，用户指定数据表以读方式(READ)锁定数据表的变体为 READ LOCAL 锁定，其与 READ 锁定的不同点是，该参数所指定的用户会话可以执行 INSERT 操作。它是为了使用 MySQL dump 工具而创建的一种变体形式。

2. 以写方式锁定数据表

与读方式锁定表类似，表的写锁定是设置用户可以修改数据表中的数据，但是除自己以外其他会话中的用户不能进行任何读操作。在命令提示符中输入如下命令。

```
lock table studentinfo write;
```

例 16.6　因为该表为写锁定，用户可以对数据库的数据执行修改、添加、删除等操作。在该命令提示符中应用 SELECT 语句查询该锁定表，其运行结果如图 16.14 所示。(**实例位置：光盘\TM\sl\16\16.6**)

```
mysql> select * from studentinfo;
+-----+--------------+-----+-----+------------+------+
| sid | name         | age | sex | tel        | time |
+-----+--------------+-----+-----+------------+------+
|   1 | LeonSK       | 21  | M   |  123456789 |      |
|   2 | HarryMason   | 22  | M   |  135791234 |      |
|   3 | BarryStephen | 22  | M   |   95135746 |      |
|   4 | JillJason    | 20  | F   |   75147631 |      |
|   5 | MrSoft       | 12  | M   |   12345678 |      |
|   6 | Chris        | 29  | M   |  654321789 |      |
|   8 | Nowitzki     | 35  | M   |   15487658 |      |
|   9 | LarryBird    | 29  | M   | 1874930203 |      |
+-----+--------------+-----+-----+------------+------+
8 rows in set (0.00 sec)
```

图 16.14　查询应用写操作锁定的数据表 studentinfo

从图 16.14 中，读者可以看到，当前用户应用 SELECT 语句仍然可以查询该表的数据，并没有限

制用户对数据表的读操作。这是因为，以写方式锁定数据表并不能限制当前锁定用户的查询操作，下面打开一个新用户会话，即保持如图 16.14 所示窗口不被关闭，重新打开一个新的 MySQL 连接，并执行上述过程。其运行结果如图 16.15 所示。

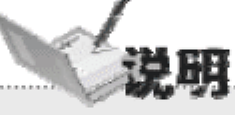
```
mysql> use students;
Database changed
mysql> select * from studentinfo;
```

图 16.15　打开新会话查询被锁定的数据表

在新打开的命令提示界面，读者可以看到，应用 SELECT 语句执行查询操作，并没有结果显示，这是因为之前该表以写方式锁定。故当操作用户释放该数据表锁定后，其他用户才可以通过 SELECT 语句查看之前被锁定的数据表。在命令提示符中输入如下代码。

```
UNLOCK TABLES;
```

这时，在第二次打开的命令提示符中，即可出现如图 16.14 所示的结果。当数据表被释放锁定后，其他访问数据库的用户即可查看数据表的内容。



使用 UNLOCK TABLE 命令后，将会释放所有当前处于锁定状态的数据表。

16.5.2　应用表锁实现伪事务

相信读者通过上面的学习，已经了解什么是表锁。下面的示例通过使用表锁对 MyISAM 表进行锁定操作，以此过程来代替事务型表 InnoDB，即应用表锁来实现伪事务。实现伪事务的一般步骤如下。

（1）对数据库中的数据表进行锁定操作，可以对多个表做不同的方式锁定，其代码格式如下。

```
LOCK TABLE table_name1 lock_type1, table_name2 lock_type2,…
```

（2）执行数据库操作，向锁定的数据表中执行添加、删除、修改操等操作。

如前面提到的 INSERT、UPDATE、DELETE 等操作。用户可以对锁定的数据表执行上述操作，在执行过程中，该伪事务所产生的结果是不会被其他用户更改的。

（3）释放锁定的数据表，以便让正在队列中等待查看或操作的其他用户可以浏览数据表中的数据或对操作表执行各种数据的操作。

如果存在其他会话要求访问已锁定的多个表格，则该会话必须被迫等待当前锁定用户释放锁定表，才允许其他会话访问该数据表，表锁定使不同会话执行的数据库操作彼此独立。应用数据表锁定方式可以使不支持事务类型的表实现伪事务。

16.6 小　　结

本章对 MySQL 中对事务的创建、提交、撤销，以及存在周期进行了详细讲解，并通过举例说明，帮助读者更好地理解所学知识的用法。在阅读本章时，读者应该重点掌握事务如何自动提交，并且修改事务的孤立级。同时读者应该了解 MySQL 事务和性能，以及 MySQL 伪事务。

16.7 实践与练习

1．在 PHP 中使用事务处理技术实现银行的安全转账。**（答案位置：光盘\TM\sl\16\16.7）**

2．在 Java 中实现模拟银行转账系统，通过事务保证转账业务的顺利进行。**（答案位置：光盘\TM\sl\16\16.8）**

第17章

事　件

在系统管理或者数据库管理中，经常要周期性地执行某一个命令或者 SQL 语句。MySQL 在 5.1 以后推出了事件调度器（Event Scheduler），可以很方便地实现 MySQL 数据库的计划任务，定期运行指定命令，使用起来非常简单和方便。

通过阅读本章，读者可以：

- 了解查看事件是否开启的方法
- 掌握开启事件的方法
- 掌握使用创建事件语句创建事件的方法
- 掌握使用修改事件语句修改事件，以及临时关闭事件的方法
- 了解删除事件的方法

17.1 事 件 概 述

在 MySQL 5.1 中新增了一个特色功能事件调度器（Event Scheduler），简称事件。它可以作为定时任务调度器，取代部分原来只能用操作系统的计划任务才能执行的工作。另外，更值得一提的是，MySQL 的事件可以实现每秒钟执行一个任务，这在一些对实时性要求较高的环境下是非常实用的。

事件调试器是定时触发执行的，从这个角度上看也可以称作是“临时触发器”。但是它与触发器又有所区别，触发器只针对某个表产生的事件执行一些语句，而事件调度器则是在某一段（间隔）时间执行一些语句。

17.1.1 查看事件是否开启

事件由一个特定的线程来管理。启用事件调度器后，拥有 SUPER 权限的账户执行 SHOW PROCESSLIST 命令就可以看到这个线程了。

例 17.1 查看事件是否开启，具体代码如下。**（实例位置：光盘\TM\sl\17\17.1）**

```
SHOW VARIABLES LIKE 'event_scheduler';
SELECT @@event_scheduler;
SHOW PROCESSLIST;
```

运行以上代码的结果如图 17.1 所示。

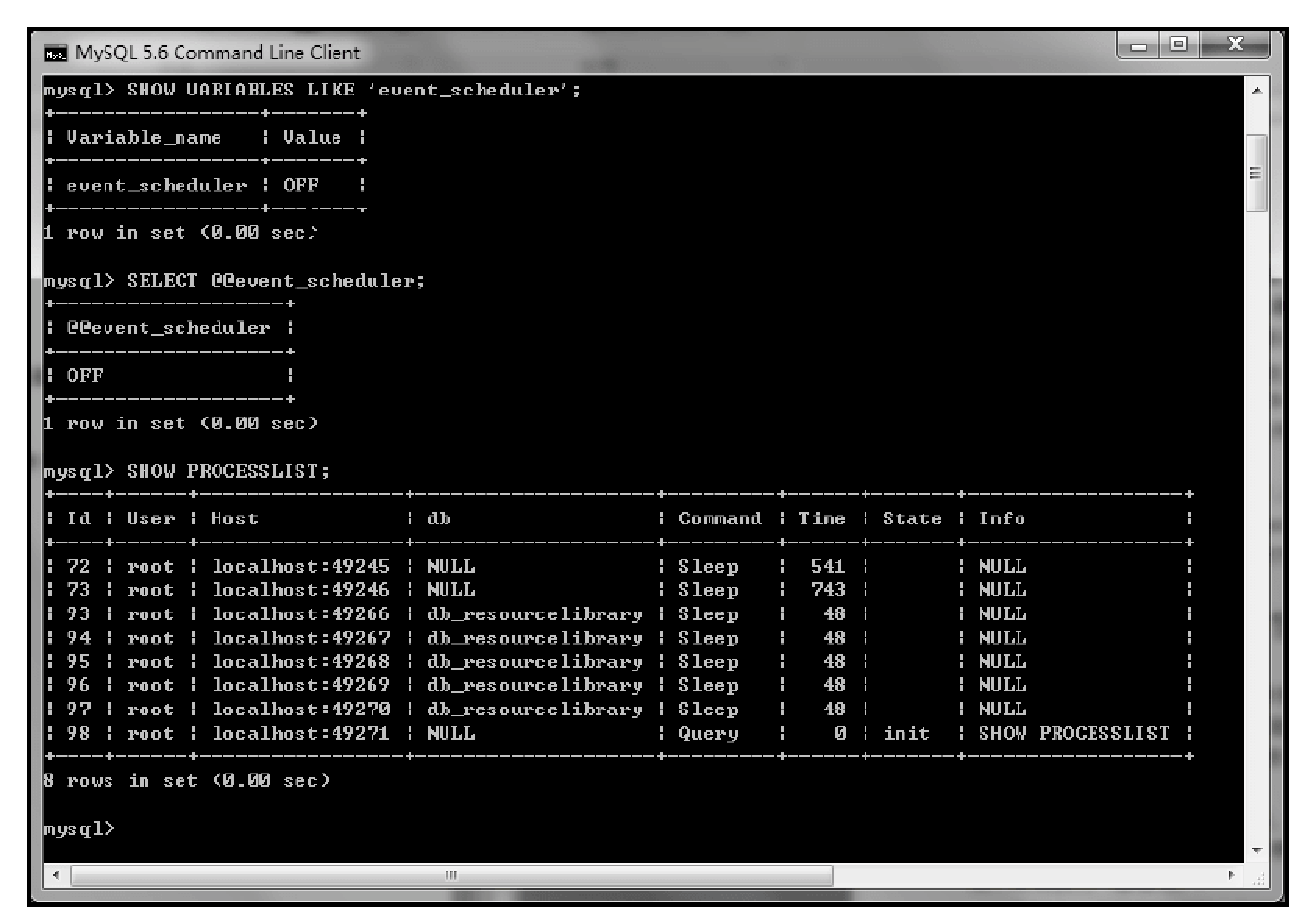


图 17.1 查看事件是否开启

从图 17.1 中可以看出事件没有开启，因为参数 event_scheduler 的值为 OFF，并且在 PROCESSLIST 中查看不到 event_scheduler 的信息；而如果参数 event_scheduler 的值为 ON，或者在 PROCESSLIST 中显示了 event_scheduler 的信息，那么就说明事件已经开启。

17.1.2　开启事件

通过设定全局变量 event_scheduler 的值即可动态地控制事件调度器是否启用。开启 MySQL 的事件调度器，可以通过下面两种方式实现。

1. 通过设置全局参数修改

在 MySQL 的命令行窗口中，使用 SET GLOBAL 命令可以开启或关闭事件。将 event_scheduler 参数的值设置为 ON，则开启事件；如果设置为 OFF，则关闭事件。例如，要开启事件可以在命令行窗口中输入下面的命令。

```
SET GLOBAL event_scheduler = ON;
```

例 17.2　开启事件并查看事件是否已经开启，具体代码如下。**（实例位置：光盘\TM\sl\17\17.2）**

```
SET GLOBAL event_scheduler = ON;
SHOW VARIABLES LIKE 'event_scheduler';
```

运行以上代码的结果如图 17.2 所示。

```
mysql> SET GLOBAL event_scheduler = ON;
Query OK, 0 rows affected (0.01 sec)

mysql> SHOW VARIABLES LIKE 'event_scheduler';
+-----------------+-------+
| Variable_name   | Value |
+-----------------+-------+
| event_scheduler | ON    |
+-----------------+-------+
1 row in set (0.00 sec)

mysql> _
```

图 17.2　开启事件并查看事件是否已经开启

从图 17.2 中，可以看出 event_scheduler 的值为 ON，则表示事件已经开启。



如果想要始终开启事件，那么在使用 SET GLOBAL 开启事件后，还需要在 my.ini/my.cnf 中添加 event_scheduler=on。因为如果没有添加，MySQL 重启事件又会回到原来的状态。

2. 更改配置文件

在 MySQL 的配置文件 my.ini（Windows 系统）/my.cnf（Linux 系统）中，找到[mysqld]，然后在下面添加以下代码开启事件。

```
event_scheduler=ON
```

在配置文件中添加代码并保存文件后，还需要重新启动 MySQL 服务器才能生效。通过该方法开启事件，重启 MySQL 服务器后，不恢复为系统默认的未开启状态。例如，此时重新连接 MySQL 服务器，然后使用下面的命令查看事件是否开启时，得到的结果将是参数 event_scheduler 的值为 ON，表示已经开启，如图 17.3 所示。

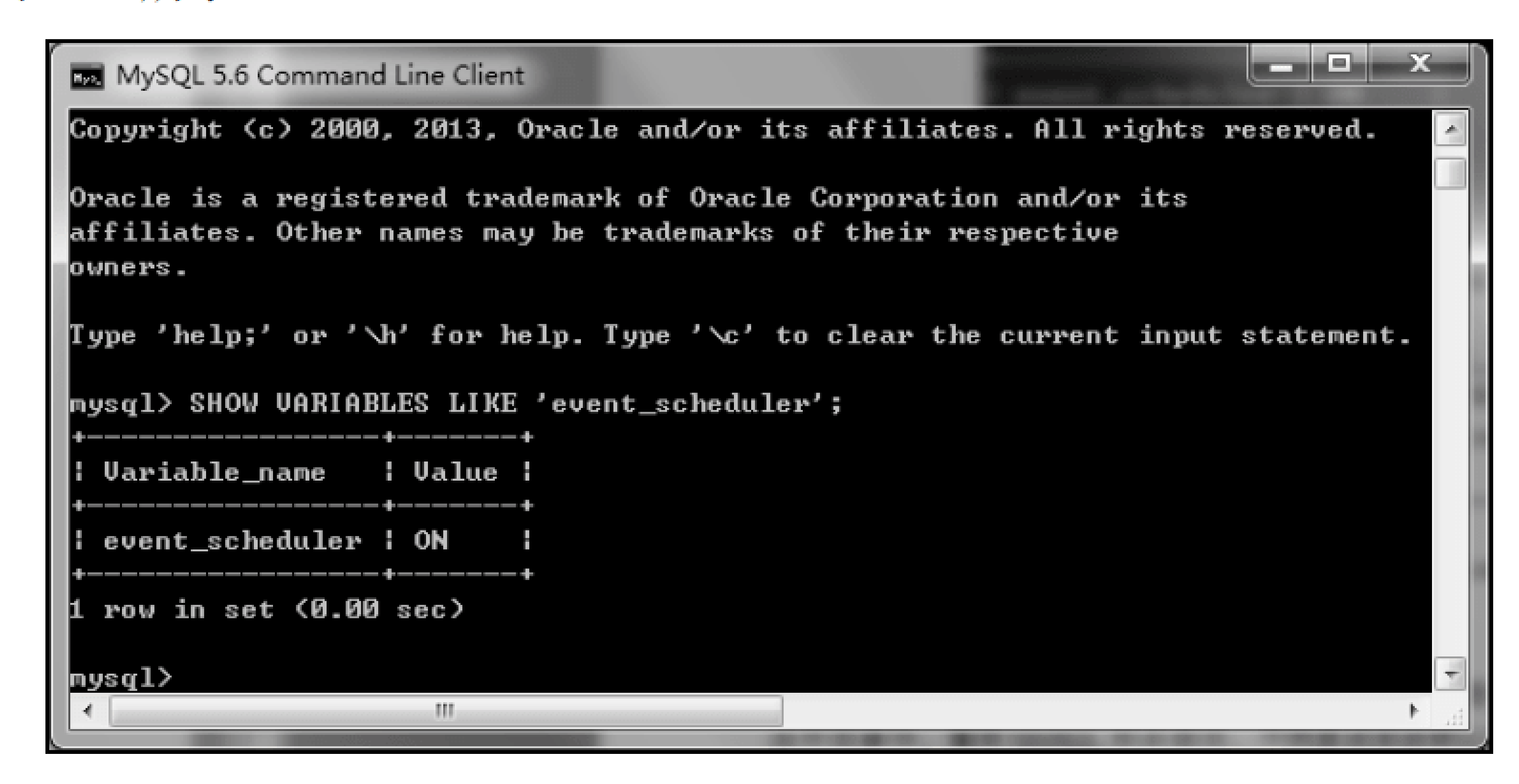


图 17.3　查看事件是否开启

17.2 创建事件

在 MySQL 5.1 以上版本中，可以通过 CREATE EVENT 语句来创建事件，其语句格式如下。

```
CREATE
    [DEFINER = { user | CURRENT_USER }]
    EVENT   [IF NOT EXISTS]   event_name
    ON SCHEDULE schedule
    [ON COMPLETION [NOT] PRESERVE]
    [ENABLE | DISABLE | DISABLE ON SLAVE]
    [COMMENT 'comment']
    DO event_body;
```

从上面的语法中，可以看出 CREATE EVENT 语句由多个子句组成，各子句的详细说明如表 17.1 所示。

表 17.1　CREATE EVENT 语句的子句

子　句	说　明
DEFINER	可选，用于定义事件执行时检查权限的用户
IF NOT EXISTS	可选，用于判断要创建的事件是否存在
EVENT event_name	必选，用于指定事件名，event_name 的最大长度为 64 个字符，如果未指定 event_name，则默认为当前的 MySQL 用户名（不区分大小写）
ON SCHEDULE schedule	必选，用于定义执行的时间和时间间隔

续表

子　　句	说　　明
ON COMPLETION [NOT] PRESERVE	可选，用于定义事件是否循环执行，即是一次执行还是永久执行，默认为一次执行，即 NOT PRESERVE
ENABLE | DISABLE | DISABLE ON SLAVE	可选，用于指定事件的一种属性。其中，关键字 ENABLE 表示该事件是活动的，也就是调度器检查事件是否必须调用；关键字 DISABLE 表示该事件是关闭的，也就是事件的声明存储到目录中，但是调度器不会检查它是否应该调用；关键字 DISABLE ON SLAVE 表示事件在从机中是关闭的。如果不指定这三个选项中的任何一个，则在一个事件创建之后，它立即变为活动的
COMMENT 'comment'	可选，用于定义事件的注释
DO event_body	必选，用于指定事件启动时所要执行的代码。可以是任何有效的 SQL 语句、存储过程或者一个计划执行的事件。如果包含多条语句，可以使用 BEGIN…END 复合结构

在 ON SCHEDULE 子句中，参数 schedule 的值为一个 AS 子句，用于指定事件在某个时刻发生，其语法格式如下。

```
    AT timestamp [+ INTERVAL interval] …
     | EVERY interval
    [STARTS timestamp [+ INTERVAL interval] …]
    [ENDS timestamp [+ INTERVAL interval] …]
```

参数说明如下。

（1）timestamp：表示一个具体的时间点，后面加上一个时间间隔，表示在这个时间间隔后事件发生。

（2）EVERY 子句：用于表示事件在指定时间区间内每隔多长时间发生一次，其中 STARTS 子句用于指定开始时间；ENDS 子句用于指定结束时间。

（3）interval：表示一个从现在开始的时间，其值由一个数值和单位构成。例如，使用“4 WEEK”表示 4 周；使用“'1:10' HOUR_MINUTE”表示 1 小时 10 分钟。间隔的距离用 DATE_ADD()函数来支配。

interval 参数值的语法格式如下。

```
quantity {YEAR | QUARTER | MONTH | DAY | HOUR | MINUTE |
                 WEEK | SECOND | YEAR_MONTH | DAY_HOUR |
                 DAY_MINUTE |DAY_SECOND | HOUR_MINUTE |
                 HOUR_SECOND | MINUTE_SECOND}
```

例 17.3　在数据库 db_database17 中创建一个名称为 e_test 的事件，用于每隔 5 秒钟向数据表 tb_eventtest 中插入一条数据。（**实例位置：光盘\TM\sl\17\17.3**）

（1）打开数据库 db_database17，代码如下。

```
use db_database17;
```

（2）创建名称为 e_test 的事件，用于每隔 5 秒钟向数据表 tb_eventtest 中插入一条数据，代码如下。

```
CREATE EVENT IF NOT EXISTS e_test ON SCHEDULE EVERY 5 SECOND
ON COMPLETION PRESERVE
DO INSERT INTO tb_eventtest(user,createtime) VALUES('root',NOW());
```

（3）创建事件后，编写以下查看数据表 tb_eventtest 中数据的代码。

```
select * from tb_eventtest;
```

执行结果如图 17.4 所示，从该图中可以看出，每隔 5 秒钟插入一条数据，这说明事件已经创建成功。

```
MySQL 5.6 Command Line Client
mysql> use db_database17;
Database changed
mysql> CREATE EVENT IF NOT EXISTS e_test ON SCHEDULE EVERY 5 SECOND
    -> ON COMPLETION PRESERVE
    -> DO INSERT INTO tb_eventtest(user,createtime) VALUES('root',NOW());
Query OK, 0 rows affected (0.15 sec)

mysql> select * from tb_eventtest;
+----+---------------------+------+
| id | createtime          | user |
+----+---------------------+------+
|  1 | 2013-12-31 09:58:25 | root |
|  2 | 2013-12-31 09:58:30 | root |
|  3 | 2013-12-31 09:58:35 | root |
|  4 | 2013-12-31 09:58:40 | root |
|  5 | 2013-12-31 09:58:45 | root |
|  6 | 2013-12-31 09:58:50 | root |
|  7 | 2013-12-31 09:58:55 | root |
|  8 | 2013-12-31 09:59:00 | root |
|  9 | 2013-12-31 09:59:05 | root |
+----+---------------------+------+
9 rows in set (0.00 sec)

mysql>
```

图 17.4　创建事件 e_test

例 17.4　创建一个事件，实现每个月的第一天凌晨 1 点统计一次已经注册的会员人数，并插入到统计表中。**（实例位置：光盘\TM\sl\17\17.4）**

（1）创建名称为 p_total 的存储过程，用于统计已经注册的会员人数，并插入到统计表 tb_total 中，具体代码如下。

```
DELIMITER //
create procedure p_total()
begin

DECLARE n_total INT default 0;
select COUNT(*) into n_total FROM db_database17.tb_user;
INSERT INTO tb_total (userNumber,createtime) values(n_total,NOW());

end
 //
```

（2）创建名称为 e_autoTotal 的事件，用于在每个月的第一天凌晨 1 点调用步骤（1）中创建的存储过程 p_total，代码如下。

```
CREATE EVENT IF NOT EXISTS e_autoTotal
```



```
 ON SCHEDULE EVERY 1 MONTH
 STARTS DATE_ADD(DATE_ADD(DATE_SUB(CURDATE(),INTERVAL DAY(CURDATE())-1 DAY),
INTERVAL 1 MONTH),INTERVAL 1 HOUR)
 ON COMPLETION PRESERVE ENABLE
 DO CALL p_total();
```

创建存储过程的执行结果如图 17.5 所示；创建事件的执行结果如图 17.6 所示。

```
mysql> use db_database17;
Database changed
mysql> DELIMITER //
mysql> create procedure p_total()
    -> begin
    ->
    -> DECLARE n_total INT default 0;
    -> select COUNT(*) into n_total FROM db_database11.tb_user;
    -> INSERT INTO tb_total (userNumber,createtime) values(n_total,NOW());
    ->
    -> end
    ->  //
Query OK, 0 rows affected (0.03 sec)

mysql>
```

图 17.5　创建存储过程 p_total

```
mysql> CREATE EVENT IF NOT EXISTS e_autoTotal
    ->  ON SCHEDULE EVERY 1 MONTH STARTS DATE_ADD(DATE_ADD(DATE_SUB(CURDATE(),INTERVAL DAY(CURDATE())-1 DAY),INTERVAL 1 MONTH),INTERVAL 1 HOUR)
    ->  ON COMPLETION PRESERVE ENABLE
    ->  DO CALL p_total();
Query OK, 0 rows affected (0.00 sec)

mysql>
```

图 17.6　创建事件 e_autoTotal

17.3　修 改 事 件

在 MySQL 5.1 及以后版本中，事件被创建之后，还可以使用 ALTER EVENT 语句修改其定义和相关属性，其语法格式如下。

```
ALTER
    [DEFINER = { user | CURRENT_USER }]
    EVENT event_name
    [ON SCHEDULE schedule]
    [ON COMPLETION [NOT] PRESERVE]
    [RENAME TO new_event_name]
    [ENABLE | DISABLE | DISABLE ON SLAVE]
    [COMMENT 'comment']
    [DO event_body]
```

ALTER EVENT 语句的使用语法与 CREATE EVENT 语句基本相同，这里不再赘述。另外，ALTER

EVENT 语句还有一个用法就是让一个事件关闭或再次让其活动。不过需要注意的是，一个事件最后一次被调用后，它是无法被修改的，因为此时它已经不存在了。

例 17.5 修改例 17.3 中创建的事件，让其每隔 30 秒向数据表 tb_eventtest 中插入一条数据。**（实例位置：光盘\TM\sl\17\17.5）**

（1）在 MySQL 的命令行窗口中，编写修改事件的代码，具体代码如下。

```
ALTER EVENT e_test ON SCHEDULE EVERY 30 SECOND
ON COMPLETION PRESERVE
DO INSERT INTO tb_eventtest(user,createtime) VALUES('root',NOW());
```

（2）编写查询数据表中数据的代码，具体代码如下。

```
SELECT * FROM tb_eventtest;
```

执行结果如图 17.7 所示。

```
mysql> ALTER EVENT e_test ON SCHEDULE EVERY 30 SECOND
    -> ON COMPLETION PRESERVE
    -> DO INSERT INTO tb_eventtest(user,createtime) VALUES('root',NOW());
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM tb_eventtest;
+----+---------------------+------+
| id | createtime          | user |
+----+---------------------+------+
| 92 | 2013-12-31 10:06:00 | root |
| 93 | 2013-12-31 10:06:05 | root |
| 94 | 2013-12-31 10:06:10 | root |
| 95 | 2013-12-31 10:06:15 | root |
| 96 | 2013-12-31 10:06:20 | root |
| 97 | 2013-12-31 10:06:21 | root |
| 98 | 2013-12-31 10:06:51 | root |
+----+---------------------+------+
98 rows in set (0.00 sec)

mysql>
```

图 17.7　修改名称为 e_test 的事件



从图 17.7 的查询结果中可以看出，在修改事件后，表 tb_eventtest 中的数据由原来的每 5 秒插入一条，改变为每 30 秒插入一条。

应用 ALTER EVENT 语句，还可以临时关闭一个已经创建的事件，下面将举例进行说明。

例 17.6 临时关闭例 17.3 中创建的事件 e_test。**（实例位置：光盘\TM\sl\17\17.6）**

（1）在 MySQL 的命令行窗口中，编写临时关闭事件 e_test 的代码，具体代码如下。

```
ALTER EVENT e_test DISABLE;
```

（2）编写查询数据表中数据的代码，具体代码如下。

```
SELECT * FROM tb_eventtest;
```

为了查看事件是否关闭，可以执行两次（每次间隔 1 分钟）步骤（2）中的代码，执行结果如图 17.8 所示。



```
mysql> ALTER EVENT e_test DISABLE;
Query OK, 0 rows affected (0.00 sec)

mysql> SELECT * FROM tb_eventtest;
+-----+---------------------+------+
| id  | createtime          | user |
+-----+---------------------+------+
|   1 | 2013-12-31 09:58:25 | root |
|   2 | 2013-12-31 09:58:30 | root |
| 127 | 2013-12-31 10:21:21 | root |
| 128 | 2013-12-31 10:21:51 | root |
+-----+---------------------+------+
4 rows in set (0.00 sec)

mysql> SELECT * FROM tb_eventtest;
+-----+---------------------+------+
| id  | createtime          | user |
+-----+---------------------+------+
|   1 | 2013-12-31 09:58:25 | root |
|   2 | 2013-12-31 09:58:30 | root |
| 127 | 2013-12-31 10:21:21 | root |
| 128 | 2013-12-31 10:21:51 | root |
+-----+---------------------+------+
4 rows in set (0.00 sec)

mysql>
```

图 17.8　临时关闭名称为 e_test 的事件



从图 17.8 的查询结果中可以看出，临时关闭事件后，将不再继续向数据表 tb_eventtest 中插入数据。

17.4　删除事件

在 MySQL 5.1 及以后版本中，删除已经创建的事件可以使用 DROP EVENT 语句来实现。DROP EVENT 语句的语法格式如下。

```
DROP EVENT [IF EXISTS] event_name
```

例 17.7　删除例 17.3 中创建的事件 e_test，代码如下。**（实例位置：光盘\TM\sl\17\17.7）**

```
use db_database17
DROP EVENT IF EXISTS e_test;
```

执行结果如图 17.9 所示。

```
mysql> use db_database17;
Database changed
mysql>
mysql> DROP EVENT IF EXISTS e_test;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

图 17.9　删除名称为 e_test 的事件

17.5 小　　结

本章首先介绍了什么是事件、如何查看事件是否开启，以及如何开启事件；然后介绍了如何创建、修改和删除事件。其中，如何开启、创建、修改和删除事件是本章的重点，需要读者认真学习并做到融会贯通，为以后的工作和学习打下良好的基础。

17.6 实践与练习

1．创建事件，实现每隔一个月清空一次新闻信息表。**（答案位置：光盘\TM\sl\17\17.8）**

2．在数据库 db_database17 中创建一个名称为 e_test 的事件，用于每隔 1 分钟向数据表 tb_eventtest 中插入一条数据。**（答案位置：光盘\TM\sl\17\17.9）**

第18章

备份与恢复

（视频讲解：3分钟）

为了保证数据的安全，需要定期对数据进行备份。备份的方式有很多种，效果也不一样。如果数据库中的数据出现了错误，就需要使用备份好的数据进行数据还原。这样可以将损失降至最低。而且，可能还会涉及数据库之间的数据导入与导出。本章将介绍备份和还原的方法，对MySQL数据库的数据安全等内容进行讲解。

通过阅读本章，读者可以：

- **掌握数据备份的使用方法**
- **掌握数据恢复的方法**
- **掌握数据库迁移的方法**
- **掌握导出和导入文本文件的方法**

18.1 数 据 备 份

备份数据是数据库管理最常用的操作。为了保证数据库中数据的安全，数据管理员需要定期进行数据备份。一旦数据库遭到破坏，即通过备份的文件来还原数据库。

可能造成数据损坏的原因很多，大体上可以归纳为以下几个方面。

（1）存储介质故障：保存数据库文件的磁盘设备损坏，同时又没有对数据库备份，从而导致数据彻底丢失。

（2）服务器彻底瘫痪：数据库服务器彻底瘫痪，系统需要重建。

（3）用户的误操作：在删除数据库时，不小心删除了某些重要数据，或者是整个数据库。

（4）黑客破坏：系统遭到黑客的恶意攻击，数据或者数据表被删除。

因此，数据备份是很重要的工作。本节将介绍数据备份的方法。

18.1.1 使用 mysqldump 命令备份

mysqldump 命令可以将数据库中的数据备份成一个文本文件。表的结构和表中的数据将存储在生成的文本文件中。本节将介绍 mysqldump 命令的工作原理和使用方法。

mysqldump 命令的工作原理很简单。它先查出需要备份的表的结构，再在文本文件中生成一条 CREATE 语句。然后，将表中的所有记录转换成一条 INSERT 语句。这些 CREATE 语句和 INSERT 语句都是还原时使用的。还原数据时就可以使用其中的 CREATE 语句来创建表。使用其中的 INSERT 语句来还原数据。

在使用 mysqldump 命令进行数据备份时，经常分为以下 3 种形式。

（1）备份一个数据库。

（2）备份多个数据库。

（3）备份所有数据库。

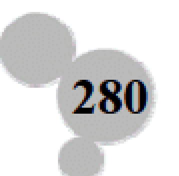
下面将分别介绍如何实现这 3 种形式的数据备份。

1. 备份一个数据库

使用 mysqldump 命令备份一个数据库的基本语法如下。

```
mysqldump –u username -p dbname table1 table2 …>BackupName.sql
```

其中，dbname 参数表示数据库的名称；table1 和 table2 参数表示表的名称，没有该参数时将备份整个数据库；BackupName.sql 参数表示备份文件的名称，文件名前面可以加上一个绝对路径。通常将数据库备份成一个后缀名为.sql 的文件。





说明

mysqldump 命令备份的文件并非一定要求后缀名为.sql，备份成其他格式的文件也是可以的，例如，后缀名为.txt 的文件。但是，通常情况下是备份成后缀名为.sql 的文件。因为，后缀名为.sql 的文件给人第一感觉就是与数据库有关的文件。

例 18.1　下面使用 root 用户备份 test 数据库下的 student 表，命令如下。**（实例位置：光盘\TM\sl\18\18.1）**

```
mysqldump –u root –p test student >D:\ student.sql
```

在 DOS 命令窗口中执行上面的命令时，将提示输入连接数据库的密码，输入密码后将完成数据备份，这时可以在 D:\中找到 student.sql 文件。student.sql 文件中的部分内容如图 18.1 所示。

```
student.sql - MR-N...XT.master (sa (51))
-- MySQL dump 10.13  Distrib 5.5.12, for
Win32 (x86)
--
-- Host: localhost    Database: test
-- ------------------------------------------
-------------
-- Server version   5.5.12

/*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@
CHARACTER_SET_CLIENT */;
/*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@
CHARACTER_SET_RESULTS */;
/*!40101 SET @OLD_COLLATION_CONNECTION=@@
COLLATION_CONNECTION */;
/*!40101 SET NAMES utf8 */;
/*!40103 SET @OLD_TIME_ZONE=@@TIME_ZONE */;
/*!40103 SET TIME_ZONE='+00:00' */;
/*!40014 SET @OLD_UNIQUE_CHECKS=@@
```

图 18.1　备份一个数据库

文件开头记录了 MySQL 的版本、备份的主机名和数据库名。文件中，以“--”开头的都是 SQL 的注释。以“/*！40101”等形式开头的内容是只有 MySQL 版本大于或等于指定的版本 4.1.1 才执行的语句。下面的“/*！40103”“/*！40014”也是这个作用。

注意

上面 student.sql 文件中没有创建数据库的语句，因此，student.sql 文件中的所有表和记录必须还原到一个已经存在的数据库中。还原数据时，CREATE TABLE 语句会在数据库中创建表，然后执行 INSERT 语句向表中插入记录。

2．备份多个数据库

mysqldump 命令备份多个数据库的语法如下。

```
mysqldump –u username –p --databases dbname1 dbname2   >BackupName.sql
```

这里要加上 databases 这个选项，然后后面跟多个数据库的名称。

例 18.2 下面使用 root 用户备份 test 数据库和 mysql 数据库，命令如下。**（实例位置：光盘\TM\sl\18\18.2）**

```
mysqldump –u root -p --databases test mysql  >D:\backup.sql
```

在 DOS 命令窗口中执行上面的命令时，将提示输入连接数据库的密码，输入密码后将完成数据备份，这时可以在 D:\下面看到名为 backup.sql 的文件，如图 18.2 所示。这个文件中存储着这两个数据库的所有信息。



图 18.2 备份多个数据库

3. 备份所有数据库

mysqldump 命令备份所有数据库的语法如下。

```
mysqldump –u username –p --all –databases >BackupName.sql
```

使用--all –databases 选项就可以备份所有数据库了。

例 18.3 下面使用 root 用户备份所有数据库。命令如下。**（实例位置：光盘\TM\sl\18\18.3）**

```
mysqldump –u root -p --all -databases  >D:\all.sql
```

在 DOS 命令窗口中执行上面的命令时，将提示输入连接数据库的密码，输入密码后将完成数据备份，这时可以在 D:\下面看到名为 all.sql 的文件，如图 18.3 所示。这个文件存储着所有数据库的所有信息。

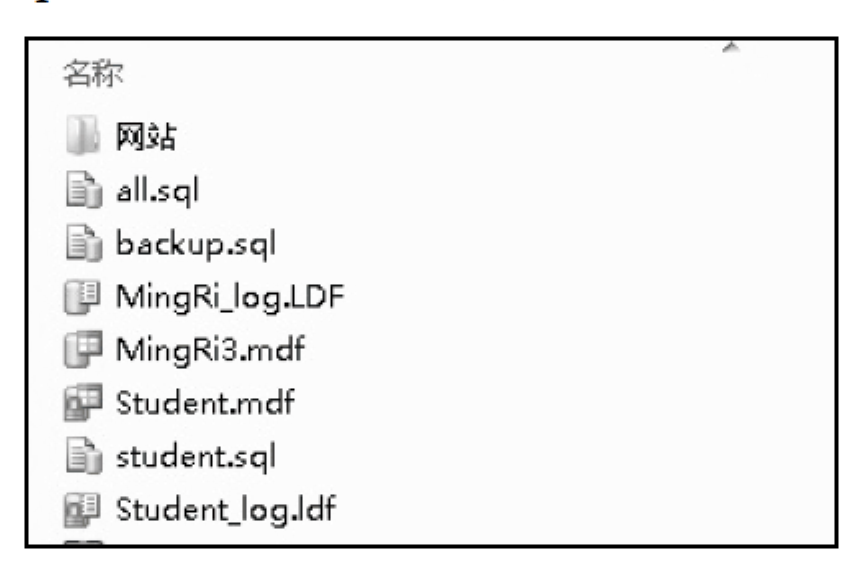


图 18.3 备份所有数据库

18.1.2 直接复制整个数据库目录

MySQL 有一种最简单的备份方法，就是将 MySQL 中的数据库文件直接复制出来。这种方法最简



单，速度也最快。使用这种方法时，最好将服务器先停止。这样，可以保证在复制期间数据库中的数据不会发生变化。如果在复制数据库的过程中还有数据写入，就会造成数据不一致。

说明

为了保证所备份数据的完整性，在停止 MySQL 服务器之前，需要先执行 FLUSH TABLES 语句将所有数据写入到数据文件的文本文件里。

这种方法虽然简单快捷，但不是最好的备份方法。因为，实际情况可能不允许停止 MySQL 服务器。而且，这种方法对 InnoDB 存储引擎的表不适用。对于 MyISAM 存储引擎的表，这样备份和还原很方便。但是还原时最好是相同版本的 MySQL 数据库，否则可能会存储文件类型不同的情况。

说明

在 MySQL 的版本号中，第一个数字表示主版本号。主版本号相同的 MySQL 数据库的文件类型会相同。例如，MySQL 5.1.39 和 MySQL 5.1.40 这两个版本的主版本号都是 5，那么这两个数据库的数据文件拥有相同的文件格式。

18.1.3 使用 mysqlhotcopy 工具快速备份

如果备份时不能停止 MySQL 服务器，可以采用 mysqlhotcopy 工具。mysqlhotcopy 工具的备份方式比 mysqldump 命令快。下面介绍 mysqlhotcopy 工具的工作原理和使用方法。

mysqlhotcopy 工具是一个 Perl 脚本，主要在 Linux 操作系统下使用。mysqlhotcopy 工具使用 LOCK TABLES、FLUSH TABLES 和 cp 来进行快速备份。其工作原理是，先将需要备份的数据库加上一个读操作锁，然后，用 FLUSH TABLES 将内存中的数据写回到硬盘上的数据库中，最后，把需要备份的数据库文件复制到目标目录。使用 mysqlhotcopy 的命令如下。

```
[root@localhost ~]#mysqlhotcopy[option] dbname1 dbname2…backupDir/
```

其中，dbname1 等表示需要备份的数据库的名称；backupDir 参数指出备份到哪个文件夹下。这个命令的含义就是将 dbname1、dbname2 等数据库备份到 backDir 目录下。mysqlhotcopy 工具有一些常用的选项，这些选项的介绍如下。

（1）--help：用来查看 mysqlhotcopy 的帮助。

（2）--allowold：如果备份目录下存在相同的备份文件，将旧的备份文件名加上_old。

（3）--keepold：如果备份目录下存在相同的备份文件，不删除旧的备份文件，而是将旧文件更名。

（4）--flushlog：本次备份之后，将对数据库的更新记录到日志中。

（5）--noindices：只备份数据文件，不备份索引文件。

（6）--user=用户名：用来指定用户名，可以用-u 代替。

（7）--password=密码：用来指定密码，可以用-p 代替。使用-p 时，密码与-p 紧挨着。或者只使用-p，然后用交换的方式输入密码。这与登录数据库时的情况是一样的。

（8）--port=端口号：用来指定访问端口，可以用-P 代替。

（9）--socket=socket 文件：用来指定 socket 文件，可以用-S 代替。

注意

mysqlhotcopy 工具不是 MySQL 自带的，需要安装 Perl 的数据接口包，Perl 的数据库接口包可以在 MySQL 官方网站下载，网址是 http://dev.mysql.com/downloads/dbi.html。mysqlhotcopy 工具的工作原理是将数据库文件复制到目标目录。因此 mysqlhotcopy 工具只能备份 MyISAM 类型的表，不能用来备份 InnoDB 类型的表。

18.2 数据恢复

管理员的非法操作和计算机的故障都会破坏数据库文件。当数据库遇到这些意外时，可以通过备份文件将数据库还原到备份时的状态。这样可以将损失降低到最小。本节将介绍数据还原的方法。

18.2.1 使用 mysql 命令还原

通常使用 mysqldump 命令将数据库的数据备份成一个文本文件。通常这个文件的后缀名是.sql。需要还原时，可以使用 mysql 命令来还原备份的数据。

备份文件中通常包含 CREATE 语句和 INSERT 语句。mysql 命令可以执行备份文件中的 CREATE 语句和 INSERT 语句。通过 CREATE 语句来创建数据库和表。通过 INSERT 语句来插入备份的数据。mysql 命令的基本语法如下。

```
mysql –uroot –p [dbname]   <backup.sql
```

其中，dbname 参数表示数据库名称。该参数是可选参数，可以指定数据库名，也可以不指定。指定数据库名时，表示还原该数据库下的表。不指定数据库名时，表示还原特定的一个数据库。备份文件中有创建数据库的语句。

例 18.4 下面使用 root 用户备份所用数据库，命令如下。**（实例位置：光盘\TM\sl\18\18.4）**

```
mysql –u root –p <D:\all.sql
```

在 DOS 命令窗口中执行上面的命令时，将提示输入连接数据库的密码，输入密码后将完成数据还原。这时，MySQL 数据库就已经还原了 all.sql 文件中的所有数据库。

注意

如果使用--all-databases 参数备份了所有的数据库，那么还原时不需要指定数据库。因为，其对应的.sql 文件包含 CREATE DATABASE 语句，可以通过该语句创建数据库。创建数据库之后，可以执行.sql 文件中的 USE 语句选择数据库，然后在数据库中创建表并且插入记录。

18.2.2 直接复制到数据库目录

之前介绍过一种直接复制数据的备份方法。通过这种方式备份的数据，可以直接复制到 MySQL 的数据库目录下。通过这种方式还原时，必须保证两个 MySQL 数据库的主版本号是相同的。而且，这种方式对 MyISAM 类型的表比较有效。对于 InnoDB 类型的表则不可用。因为 InnoDB 表的表空间不能直接复制。

在 Windows 操作系统下，MySQL 的数据库目录通常存放在下面 3 个路径的其中之一，分别是 C:\mysql\date、C:\Documents and Settings\All Users\Application Data\MySQL\MySQL Server5.1\data 或者 C:\Program Files\MySQL Server 5.1\data。在 Linux 操作系统下，数据库目录通常在/var/lib/mysql/、/usr/local/mysql/data 或者/usr/local/mysql/var 这 3 个目录下。上述位置只是数据库目录最常用的位置。具体位置根据读者安装时设置的位置而定。

使用 mysqlhotcopy 命令备份的数据也是通过这种方式来还原的。在 Linux 操作系统下，复制到数据库目录后，一定要将数据库的用户和组变成 mysql，命令如下。

```
chown –R mysql.mysql dataDir
```

其中，两个 mysql 分别表示组和用户；-R 参数可以改变文件夹下的所有子文件的用户和组；dataDir 参数表示数据库目录。

注意

Linux 操作系统下的权限设置非常严格。通常情况下，MySQL 数据库只有 root 用户和 mysql 用户组下的 mysql 用户可以访问。因此，将数据库目录复制到指定文件夹后，一定要使用 chown 命令将文件夹的用户组变为 mysql，将用户变为 mysql。

18.3 数据库迁移

数据库迁移就是指将数据库从一个系统移动到另一个系统上。数据库迁移的原因是多种多样的。可能是因为升级了计算机，或者是部署开发的管理系统，或者是升级了 MySQL 数据库，甚至是换用其他的数据库。根据上述情况，可以将数据库迁移大致分为 3 类，分别是在相同版本的 MySQL 数据库之间迁移、迁移到其他版本的 MySQL 数据库中和迁移到其他类型的数据库中。本节将介绍数据库迁移的方法。

18.3.1 相同版本的 MySQL 数据库之间的迁移

相同版本的 MySQL 数据库之间的迁移就是在主版本号相同的 MySQL 数据库之间进行数据库移

动。这种迁移的方式最容易实现。本节将介绍这方面的内容。

相同版本的 MySQL 数据库之间进行数据库迁移的原因很多。通常的原因是换了新的机器，或者是装了新的操作系统。还有一种常见的原因就是将开发的管理系统部署到工作机器上。因为迁移前后 MySQL 数据库的主本版号相同，所以可以通过复制数据库目录来实现数据库迁移。但是，只有数据库表都是 MyISAM 类型的才能使用这种方式。

最常用和最安全的方式是使用 mysqldump 命令来备份数据库。然后使用 mysql 命令将备份文件还原到新的 MySQL 数据库中。这里可以将备份和迁移同时进行。假设从一个名为 host1 的机器中备份出所有数据库，然后，将这些数据库迁移到名为 host2 的机器上。命令如下。

```
mysqldump –h name1 –u root –password=password1 –all-databases |
mysql –h host2 –u root –password=password2
```

其中，“|”符号表示管道，其作用是将 mysqldump 备份的文件送给 mysql 命令；“-password=password1”是 name1 主机上 root 用户的密码；同理，password2 是 name2 主机上的 root 用户的密码。通过这种方式可以直接实现迁移。

18.3.2　不同数据库之间的迁移

不同数据库之间迁移是指从其他类型的数据库迁移到 MySQL 数据库，或者从 MySQL 数据库迁移到其他类型的数据库。例如，某个网站原来使用 Oracle 数据库，因为运营成本太高等诸多原因，希望改用 MySQL 数据库；或者，某个管理系统原来使用 MySQL 数据库，因为某种特殊性能的要求，希望改用 Oracle 数据库。这样的不同数据库之间的迁移也经常会发生，但是这种迁移没有普通适用的解决方法。

MySQL 以外的数据库也有类似 mysqldump 这样的备份工具，可以将数据库中的文件备份成.sql 文件或普通文件。但是，因为不同数据库厂商没有完全按照 SQL 标准来设计数据库，这就造成了不同数据库使用的 SQL 语句的差异。例如，微软的 SQL Server 软件使用的是 T-SQL。T-SQL 中包含非标准的 SQL 语句。这就造成了 SQL Server 和 MySQL 的 SQL 语句不能兼容。

除了 SQL 语句存在不兼容的情况下，不同的数据库之间的数据类型也有差异。数据类型的差异也造成了迁移的困难。例如，SQL Server 数据库中有 ntext、Image 等数据类型，而 MySQL 数据库都没有；MySQL 支持的 ENUM 和 SET 类型，这些 SQL Server 数据库不支持。从某种意义上说，这种差异是商业数据库公司故意造成的壁垒，是阻碍数据库市场健康发展的。

18.4　表的导出和导入

MySQL 数据库中的表可以导出成文本文件、XML 文件或者 HTML 文件，相应的文本文件也可以导入 MySQL 数据库中。在数据库的日常维护中，经常需要进行表的导出和导入的操作。本节将介绍将表内容导出和导入文本文件的方法。

18.4.1　用 SELECT …INTO OUTFILE 导出文本文件

MySQL 中，可以在命令行窗口（MySQL Commend Line Client）中使用 SELECT…INTO OUTFILE 语句将表的内容导出成一个文本文件。其基本语法形式如下。

```
SELECT[列名] FROM table[WHERE 语句]
INTO OUTFILE '目标文件'[OPTION];
```

该语句分为两个部分。前半部分是一个普通的 SELECT 语句，通过这个 SELECT 语句来查询所需要的数据；后半部分是导出数据的。其中，“目标文件”参数指出将查询的记录导出到哪个文件；OPTION 参数可以有常用的几个选项。介绍如下。

（1）FIELDS TERMINATED BY'字符串'：设置字符串为字段的分隔符，默认值是“\t”。

（2）FIELDS ENCLOSED BY'字符'：设置字符来括上字段的值。默认情况下不使用任何符号。

（3）FIELDS OPTIOINALLY ENCLOSED BY'字符'：设置字符来括上 CHAR、VARCHAR 和 TEXT 等字符型字段。默认情况下不使用任何符号。

（4）FIELDS ESCAPED BY'字符'：设置转义字符，默认值为“\”。

（5）LINES STARTING BY'字符串'：设置每行开头的字符，默认情况下无任何字符。

（6）LINES TERMINATED BY'字符串'：设置每行的结束符，默认值是“\n”。

例 18.5　下面用 SELECT…INTO OUTFILE 语句来导出 test 数据库下 order 表的记录。其中，字段之间用“、”隔开，字符型数据用双引号括起来，每条记录以“>”开头。命令如下。（**实例位置：光盘\TM\sl\18\18.5**）

```
SELECT * FROM test.order INTO OUTFILE 'D:\order.txt'
FIELDS TERMINATED BY '\、' OPTIONALLY ENCLOSED BY '\"'
LINES STARTING BY '\>' TERMINATED BY '\r\n';
```

“TERMINATED BY '\r\n'”可以保证每条记录占一行。因为 Windows 操作系统下“\r\n”才是回车换行。如果不加这个选项，默认情况只是“\n”。用 root 用户登录到 MySQL 数据库中，然后执行上述命令。执行完后，可以在 D:\下看到一个名为 order.txt 的文本文件。order.txt 中的内容如图 18.4 所示。

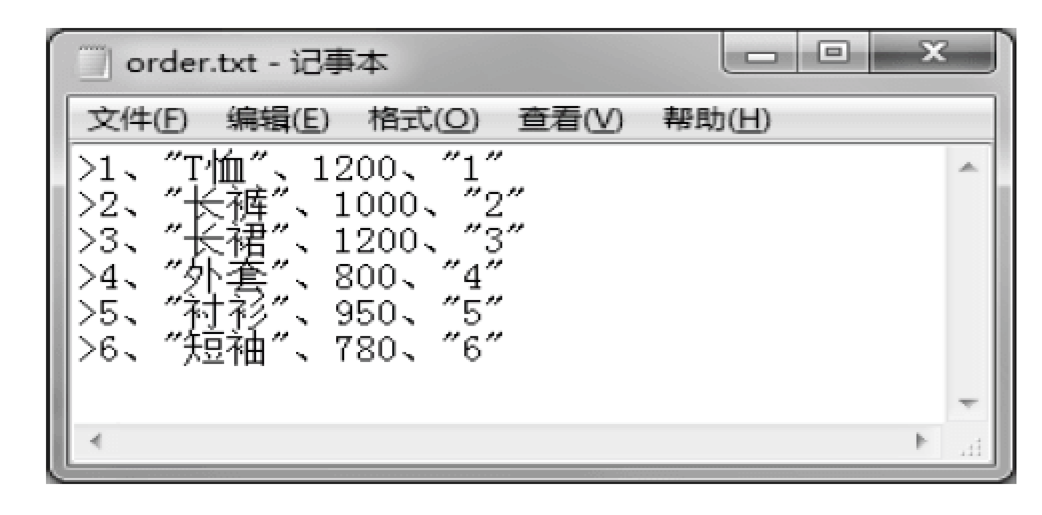


图 18.4　用 SELECT…INTO OUTFILE 导出文本文件

这些记录都是以“>”开头，每个字段之间以“、”隔开。而且，字符数据都加上了引号。

18.4.2 用 mysqldump 命令导出文本文件

mysqldump 命令可以备份数据库中的数据。但是，备份时是在备份文件中保存了 CREATE 语句和 INSERT 语句。不仅如此，mysqldump 命令还可以导出文本文件。其基本的语法形式如下。

```
mysqldump –u root –pPassword –T 目标目录 dbname table [option];
```

其中，Password 参数表示 root 用户的密码，密码紧挨着-p 选项；目标目录参数指出文本文件的路径；dbname 参数表示数据库的名称；table 参数表示表的名称；option 表示附件选项。这些选项介绍如下。

（1）--fields-terminated-by=字符串：设置字符串为字段的分隔符，默认值是“\t”。

（2）--fields-enclosed-by=字符：设置字符来括上字段的值。

（3）--fields-optionally-enclosed-by=字符：设置字符括上 CHAR、VARCHAR 和 TEXT 等字符型字段。

（4）--fields-escaped-by=字符：设置转义字符。

（5）--lines-terminated-by=字符串：设置每行的结束符。



这些选项必须用双引号括起来，否则，MySQL 数据库系统将不能识别这几个参数。

例 18.6 用 mysqldump 语句来导出 test 数据库下 order 表的记录。其中，字段之间用“、”隔开，字符型数据用双引号括起来。命令如下。**（实例位置：光盘\TM\sl\18\18.6）**

```
mysqldump -u root -p -T D:\ test order "--lines-terminated-by=\r\n"
"--fields-terminated-by=、" "--fields-optionally-enclosed-by=""
```

其中，root 用户的密码为 111，密码紧挨着-p 选项。--fields-terminated-by 等选项都用双引号括起来。命令执行完后，可以在 D:\下看到一个名为 order.txt 的文本文件和 order.sql 文件。order.txt 中的内容如图 18.5 所示。

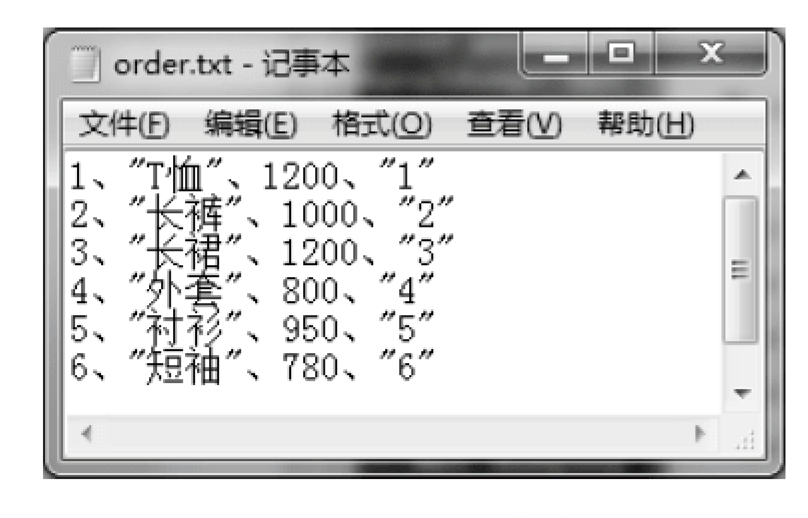
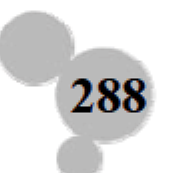


图 18.5 用 MYSQLDUMP 命令导出文本文件

这些记录都是以“、”隔开，而且，字符数据都是加上了引号。其实，mysqldump 命令也是调用 SELECT…INTO OUTFILE 语句来导出文本文件的；同时 mysqldump 命令还生成了 student.sql 文件。这



个文件中有表的结构和表中的记录。



导出数据时，一定要注意数据的格式。通常每个字段之间都必须用分隔符隔开，可以使用逗号（,）、空格或者制表符（Tab 键）。每条记录占用一行，新记录要从下一行开始。字符串数据要使用双引号括起来。

mysqldump 命令还可以导出 XML 格式的文件，其基本语法如下。

```
mysqldump-u root –pPassword --xml|–X dbname table >D:\name.xml;
```

其中，Password 表示 root 用户的密码；使用--xml 或者-X 选项就可以导出 XML 格式的文件；dbname 表示数据库的名称；table 表示表的名称；D:\name.xml 表示导出的 XML 文件的路径。

例 18.7　使用 mysqldump 命令将数据表 student 中的内容导出到 XML 文件中。效果如图 18.6 所示。**（实例位置：光盘\TM\sl\18\18.7）**

```
C:\Users\Administrator>mysqldump -u root -p --xml test student >D:/student.xml
Enter password: ***

C:\Users\Administrator>
```

图 18.6　在 DOS 命令窗口中的执行效果

生成的 XML 文件可以在 D 盘的根目录下找到，内容如图 18.7 所示。

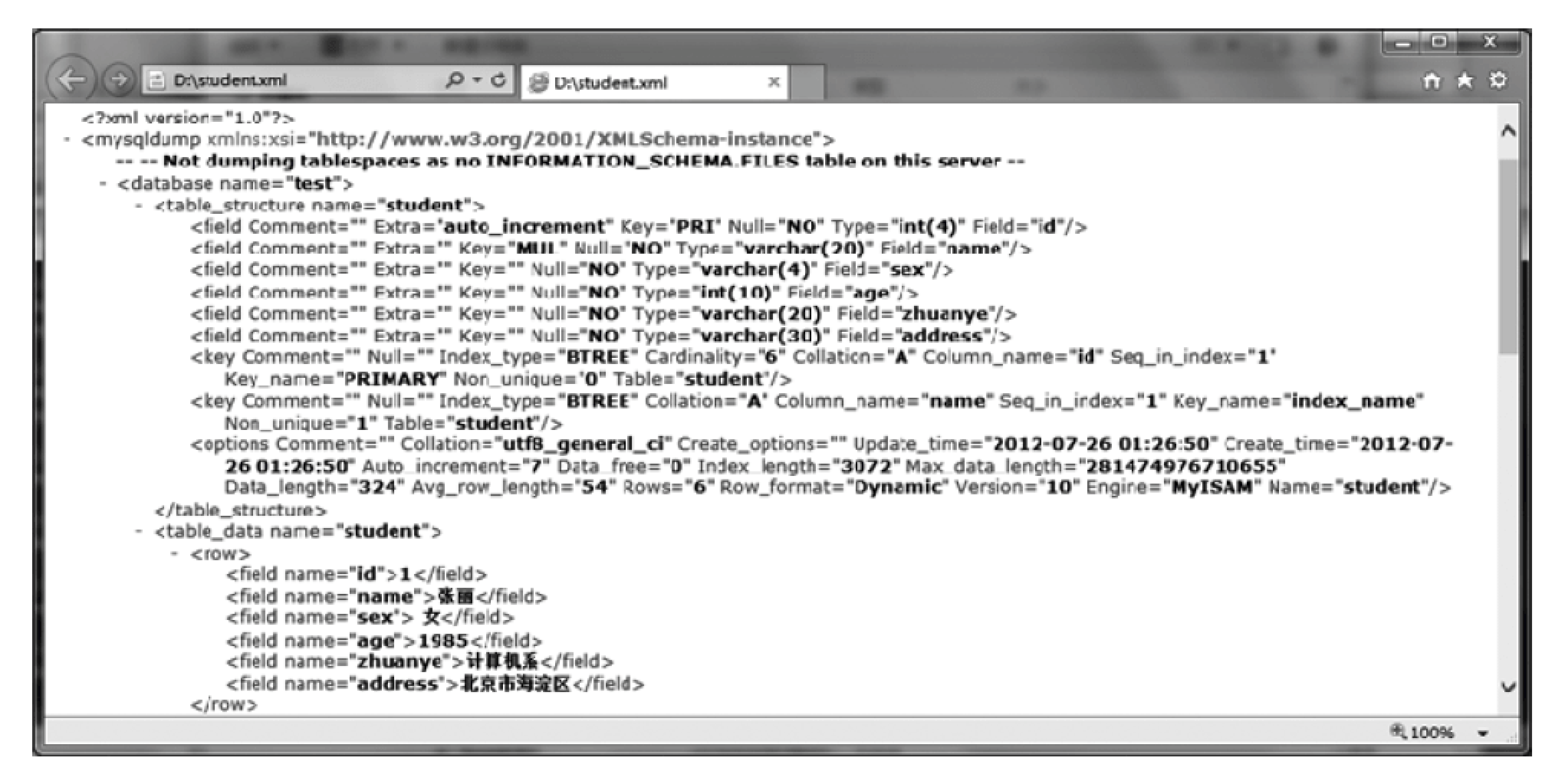


图 18.7　生成的 XML 文件

18.4.3　用 mysql 命令导出文本文件

mysql 命令可以用来登录 MySQL 服务器，也可以用来还原备份文件。同时，mysql 命令也可以导出文本文件。其基本语法形式如下。

```
mysql –u root –pPassword –e "SELECT  语句" dbname >D:/name.txt;
```

其中，Password 表示 root 用户的密码；使用-e 选项就可以执行 SQL 语句：“SELECT 语句”用来查询记录；D:/name.txt 表示导出文件的路径。

例 18.8 下面用 mysql 命令来导出 test 数据库下 student 表的记录，命令如下。（**实例位置：光盘\TM\sl\18\18.8**）

```
mysql -u root -p111 -e"SELECT * FROM student" test > D:/student2.txt
```

在 DOS 命令窗口中执行上述命令，可以将 student 表中的所用记录查询出来，然后写入到 student2.txt 文档中。student2.txt 被保存在 D 盘根目录下，如图 18.8 所示。

student2.txt 中的内容如图 18.9 所示。



图 18.8 mysql 命令导出文本文件



id	name	sex	age	zhuanye	address
1	张丽	女	1985	计算机系	北京市海淀区
2	李小燕	女	1986	中文系	北京市昌平区
3	张三	男	1990	中文系	湖南省永州市
4	李四	男	1990	英语系	辽宁省阜新市
5	王五	女	1991	英语系	福建省厦门市
6	章也	男	1988	计算机系	湖南省衡阳市

图 18.9 文档内容

mysql 命令还可以导出 XML 文件和 HTML 文件。mysql 命令导出 XML 文件的语法如下。

```
mysql –u root –pPassword --xml|–X –e "SELECT  语句" dbname >D:/filename.xml
```

其中，Password 表示 root 用户的密码；使用--xml 或者-X 选项就可以导出 XML 格式的文件；dbname 表示数据库的名称；D:/name.xml 表示导出的 XML 文件的路径。

例如，下面的命令可以将 test 数据库中的 student 表的数据导出到名称为 student.xml 的 XML 文件中。

```
mysql -u root -p111 --xml   -e "SELECT * from student" test >D:/ student.xml
```

mysql 命令导出 HTML 文件的语法如下。

```
mysql –u root –pPassword --html|–H –e "SELECT  语句" dbname >D:/filename.html
```

其中，使用--html 或者-H 选项就可以导出 HTML 格式的文件。

例如，下面的命令可以将 test 数据库中的 student 表的数据导出到名称为 student.html 的 HTML 文件中。

```
mysql -u root -p111 --html   -e "SELECT * from student" test >D:/student.html
```

18.4.4 用 LOAD DATA INFILE 命令将文本文件导入到数据表

在 MySQL 中，可以通过命令 LOAD DATA INFILE 来实现将指定格式的文本文件导入到数据表中。



LOAD DATA INFILE 命令的语法格式如下。

```
LOAD DATA [LOW_PRIORITY|CONCURRENT] [LOCAL] INFILE file_name INTO TABLE table_name [OPTION];
```

参数说明如下。

（1）LOW_PRIORITY：如果指定 LOW_PRIORITY，则 LOAD DATA 语句会被延迟，直到没有其他的客户端正在读取表。

（2）CONCURRENT：如果指定 CONCURRENT，则当 LOAD DATA 正在执行时，其他线程可以同时使用该表的数据。

（3）LOCAL：如果指定了 LOCAL，则文件会被客户主机上的客户端读取，并被发送到服务器。文件会被给予一个完整的路径名称，以指定其确切的位置。如果给定的是一个相对路径名称，则此名称会被理解为相对于启动客户端时所在的目录。如果没有指定 LOCAL，则文件必须位于服务器主机上，并且被服务器直接读取。使用 LOCAL 的速度会略慢些。

（4）file_name：用来指定要导入的文本文件的路径和名称。这个文件可以手动创建，也可以使用其他的程序创建。可以使用绝对路径（如 D:/student1.txt），也可以不指定路径，直接写上文件名（如 student1.txt），这时服务器将在默认数据库目录中查找并读取。

（5）table_name 用来指定需要导入数据的表名，该表在数据库中必须存在，表结构必须与导入文件的数据一致。

（6）OPTION 用于设置相应的选项，其值可以是下面 9 个值中的任何一个。

① FIELDS TERMINATED BY '字符串'：用于设置字段的隔符为字符串对象，默认值为“\t”。

② FIELDS ENCLOSED BY '字符'：用于设置括上字段值的字符符号，默认情况下不使用任何符号。

③ FIELDS OPTIONALLY ENCLOSED BY 字符：用来设置括上 CHAR、VARCHAR 和 TEXT 等字段值的字符符号，默认情况下不使用任何符号。

④ FIELDS ESCAPED BY ''：用于设置转义字符的字符符号，默认情况下使用“\”字符。

⑤ LINES STARTING BY '字符'：用来设置每行开头的字符符号，默认情况下不使用任何符号。

⑥ LINES TERMINATED BY '字符串'：用于设置每行结束的字符串符号，默认情况下使用“\n”字符串。

⑦ IGNORE n LINES：用于指定忽略文件的前 n 行记录。

⑧（字段列表）：用于实现根据字段列表中的字符和顺序来加载记录。

⑨ SET column=expr：用于设置列的转换条件，即所指定的列经过相应转换后才会被加载。

注意

在使用该命令时，必须根据要导入文本文件中，字段值的分隔符来指定使用的分隔符；并且如果文本文件中字段的顺序与数据表中字段的顺序不完全一致，就需要使用“（字段列表）”来指定加载字段的顺序。

例 18.9　使用 LOAD DATA INFILE 命令，将 D 盘根目录下的 student1.txt 文件中的数据记录导入到数据库 test 中的 student1 表中。**（实例位置：光盘\TM\sl\18\18.9）**

（1）准备一个名称为 student1.txt 的文本文件，并放置在 D 盘根目录下，内容如图 18.10 所示，对

于这个文件，可以使用例 18.8 介绍的方法来进行导出。

（2）进入到 MySQL 的命令行窗口中，输入以下命令选择数据库 test。

```
USE test;
```

执行效果如图 18.11 所示。



图 18.10　student1.txt 的内容

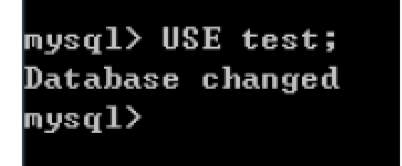


图 18.11　选择数据库

（3）在 test 数据库中，创建一张名称为 student1 的数据表，表结构如图 18.12 所示。

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	Default
id	INT(4)	☑	☑	☐	☐	☐	☐	☑	
name	VARCHAR(20)	☐	☑	☐	☐	☐	☐	☐	
sex	VARCHAR(4)	☐	☑	☐	☐	☐	☐	☐	
age	INT(10)	☐	☑	☐	☐	☐	☐	☐	
zhuanye	VARCHAR(20)	☐	☑	☐	☐	☐	☐	☐	
address	VARCHAR(30)	☐	☑	☐	☐	☐	☐	☐	
		☐	☐	☐	☐	☐	☐	☐	

图 18.12　数据表 student1 的表结构

（4）执行 LOAD DATA INFILE 命令，将文本文件 student1.txt 中的数据导入到表 student1 中，具体代码如下。

```
LOAD DATA INFILE 'D:/student1.txt' INTO TABLE student1
 FIELDS TERMINATED BY '\t'
 LINES TERMINATED BY '\r\n'
 IGNORE 1 LINES;
```

执行效果如图 18.13 所示，数据表 student1 中的数据如图 18.14 所示。

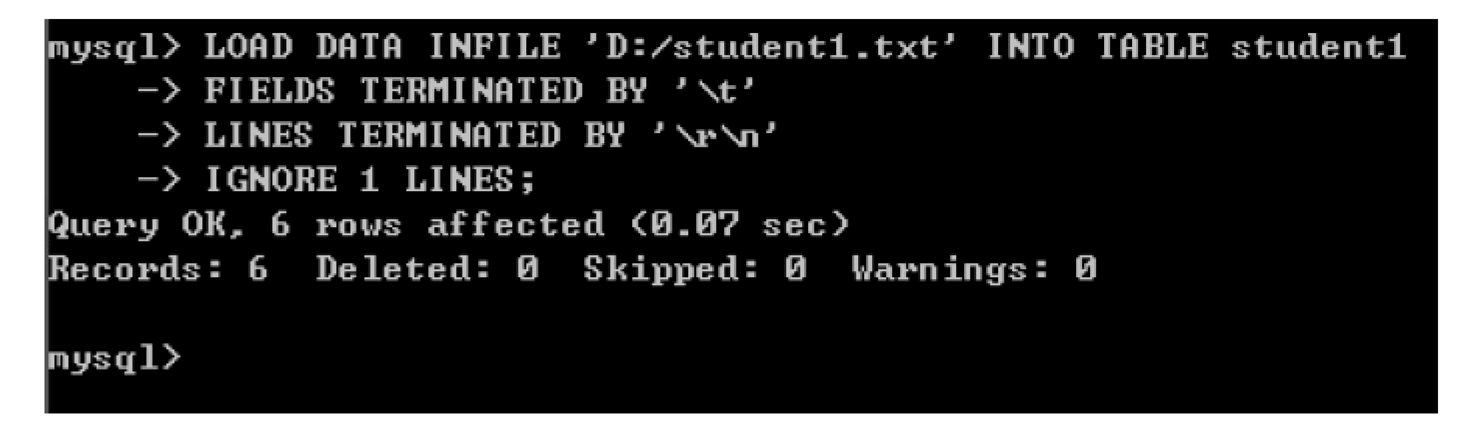


图 18.13　执行 LOAD DATA INFILE 命令导入文本文件到数据表

（5）应用 SELECT 语句查询数据表 student1 中的数据，代码如下。

```
select * from student1;
```

执行结果如图 18.14 所示。



```
mysql> select * from student1;
+----+--------+-----+------+----------+----------------+
| id | name   | sex | age  | zhuanye  | address        |
+----+--------+-----+------+----------+----------------+
|  1 | 李丽   |  女 | 1985 | 计算机系 | 北京市海淀区   |
|  2 | 李小燕 | 女  | 1986 | 中文系   | 北京市昌平区   |
|  3 | 张三   | 男  | 1990 | 中文系   | 湖南省永州市   |
|  4 | 李四   | 男  | 1990 | 英语系   | 辽宁省阜新市   |
|  5 | 王五   | 女  | 1991 | 英语系   | 福建省厦门市   |
|  6 | 章也   | 男  | 1988 | 计算机系 | 吉林省长春市   |
+----+--------+-----+------+----------+----------------+
6 rows in set (0.00 sec)

mysql>
```

图 18.14　数据表 student1 的数据

18.4.5　用 mysqlimport 命令导入文本文件

在 MySQL 中，如果只是恢复数据表中的数据，可以在 Windows 的命令提示符窗口中使用 mysqllimport 命令来实现。通过 mysqlimport 命令可以实现将指定格式的文本文件导入到数据表中。实际上，这个命令提供了 LOAD DATA INFILE 语句的一个命令行接口，它发送一个 LOAD DATA INFILE 命令到服务器来运行，它的大多数选项直接对应于 LOAD DATA INFILE 命令。mysqlimport 命令的语法格式如下。

```
mysqlimport --no-defaults –u root –pPassword –T database file_name [option];
```


注意

这里的-p 与 Password 之间是没有空格的。

其中，--no-defaults 表示指定不要从任何选项文件中读取默认选项，这是必选项；Password 参数表示 root 用户的密码，密码紧挨着-p 选项；目标目录参数指出文本文件的路径；database 参数表示要导入数据的数据库的名称；file_name 参数表示要导入数据的文本文件名；OPTION 用于设置相应的选项，其值可以是下面几个值中的任何一个。

（1）--fields-terminated-by=字符串：用于设置字段的分隔符，默认值是“\t”。

（2）--fields-enclosed-by=字符：用于设置括上字段值的字符符号，默认情况下不使用任何符号。

（3）--fields-optionally-enclosed-by=字符：用于设置括上 CHAR、VARCHAR 和 TEXT 等字符型字段值的字符符号，默认情况下不使用任何符号。

（4）--fields-escaped-by=字符：用于设置转义字符。

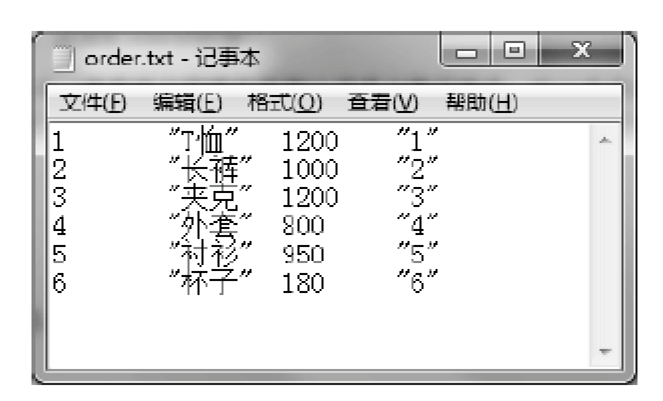
（5）--lines-terminated-by=字符串：用于设置每行的结束符。

（6）--ignore-lines=n：用于指定忽略文件的前 n 行记录。

例 18.10　使用 mysqlimport 命令，将 D 盘根目录下的 student1.txt 文件中的数据记录导入到数据库 test 中的 student1 表中。**（实例位置：光盘\TM\sl\18\18.10）**

（1）准备一个名称为 order.txt 的文本文件，并放置在 D 盘根目录下，内容如图 18.15 所示。

（2）在 test 数据库中，创建一张名称为 order 的数据表，表结构如图 18.16 所示。


order.txt - 记事本
文件(F) 编辑(E) 格式(O) 查看(V) 帮助(H)

```
1	"T恤"	1200	"1"
2	"长裤"	1000	"2"
3	"夹克"	1200	"3"
4	"外套"	800	"4"
5	"衬衫"	950	"5"
6	"杯子"	180	"6"
```

图 18.15 order.txt 的内容

Column Name	Datatype	PK	NN	UQ	BIN	UN	ZF	AI	Default
oid	INT(11)	☑	☑	☐	☐	☐	☐	☐	
proname	VARCHAR(45)	☐	☐	☐	☐	☐	☐	☐	NULL
procount	INT(11)	☐	☐	☐	☐	☐	☐	☐	NULL
cid	VARCHAR(45)	☐	☐	☐	☐	☐	☐	☐	NULL

图 18.16 数据表 order 的表结构

（3）执行 mysqlimport 命令，将文本文件 order.txt 中的数据导入到表 order 中，具体代码如下。

```
mysqlimport --no-defaults -u root -p111 test D:\order.txt "--lines-terminated-by=\r\n" "--fields-terminated-by=\t"
 "--fields-optionally-enclosed-by=\""
```

执行效果如图 18.17 所示。

```
C:\Users\Administrator>mysqlimport --no-defaults -u root -p111 test D:\order.txt "--lines-terminated-by=\r\n" "--fields-terminated-by=\t" "--fields-op
tionally-enclosed-by=\""
Warning: Using a password on the command line interface can be insecure.
test.order: Records: 6  Deleted: 0  Skipped: 0  Warnings: 0

C:\Users\Administrator>
```

图 18.17 执行 mysqlimport 命令导入文本文件到数据表

（4）应用 SELECT 语句查询数据表 order 中的数据，代码如下。

```
use test;
select * from `order`;
```

执行结果如图 18.18 所示。

```
mysql> use test;
Database changed
mysql> select * from `order`;
+-----+---------+----------+------+
| oid | proname | procount | cid  |
+-----+---------+----------+------+
|   1 | 杯子    |     1200 | 1    |
|   2 | 衬衫    |     1000 | 2    |
|   3 | 外套    |     1200 | 3    |
|   4 | 夹克    |      800 | 4    |
|   5 | 长裤    |      950 | 5    |
|   6 | T恤     |      780 | 6    |
+-----+---------+----------+------+
6 rows in set (0.00 sec)

mysql>
```

图 18.18 数据表 order 的数据



在 MySQL 中，表名和字符串可以用反引号（也就是数字 1 左侧的那个键）括起来，但是这并不是必需的，不过，如果使用的表名或者字段是 MySQL 中的关键字，那么使用反引号将其括起来就是必需的。

18.5　小　　结

本章对备份数据库、还原数据库、数据库迁移、导出表和导入表进行了详细讲解，备份数据库和还原数据库是本章的重点内容。在实际应用中，通常使用 mysqldump 命令备份数据库，使用 mysql 命令还原数据库。数据库迁移、导出表和导入表是本章的难点。数据迁移需要考虑数据库的兼容性问题，最好是在相同版本的 MySQL 数据库之间迁移。导出表和导入表的方法比较多，希望读者能够多练习这些方法的使用。

18.6 实践与练习

1．实现将数据表 student 中的内容导出到文本文件中，在生成文本文件时，每个字段之间用逗号隔开。每个字符型的数据用双引号括起来。而且，每条记录占一行。**（答案位置：光盘\TM\sl\18\18.11）**

2．使用 mysql 命令，将表中的记录导出到 HTML 文件中。**（答案位置：光盘\TM\sl\18\18.12）**

第 19 章

MySQL 性能优化

（视频讲解：10 分钟）

性能优化是通过某些有效的方法提高 MySQL 数据库的性能。性能优化的目的是为了使 MySQL 数据运行速度更快、占用的磁盘空间更小。性能优化包括很多方面，例如优化查询速度、优化更新速度和优化 MySQL 服务器等。本章将介绍性能优化的目的，优化查询、优化数据库结构和优化多表查询等的方法，以提高 MySQL 数据库的速度。

通过阅读本章，读者可以：

- 了解优化查询的方法
- 了解优化数据库结构的方法
- 了解查询高速缓存的方法
- 了解优化多表查询的方法
- 了解优化表设计的方法

19.1　优 化 概 述

优化 MySQL 数据库是数据库管理员的必备技能，通过不同的优化方式达到提高 MySQL 数据库性能的目的。本节将介绍优化的基本知识。

MySQL 数据库的用户和数据非常少的时候，很难判断一个 MySQL 数据库性能的好坏。只有当长时间运行，并且有大量用户进行频繁操作时，MySQL 数据库的性能才能体现出来。例如，一个每天有几万用户同时在线的大型网站的数据库性能的优劣就很明显。这么多用户在同时连接 MySQL 数据库，并且进行查询、插入和更新的操作，如果 MySQL 数据库的性能很差，很可能无法承受如此多用户同时操作。试想如果用户查询一条记录需要花费很长时间，用户很难会喜欢这个网站。

因此，为了提高 MySQL 数据库的性能，需要进行一系列的优化措施。如果 MySQL 数据库需要进行大量的查询操作，那么就需要对查询语句进行优化。对于耗费时间的查询语句进行优化，可以提高整体的查询速度。如果连接 MySQL 数据库用户很多，那么就需要对 MySQL 服务器进行优化；否则，大量的用户同时连接 MySQL 数据库，可能会造成数据库系统崩溃。

数据库管理员可以使用 SHOW STATUS 语句查询 MySQL 数据库的性能。语法形式如下。

```
SHOW STATUS LIKE 'value';
```

其中，value 参数是常用的几个统计参数，具体介绍如下。

（1）Connections：连接 MySQL 服务器的次数。

（2）Uptime：MySQL 服务器的上线时间。

（3）Slow_queries：慢查询的次数。

（4）Com_select：查询操作的次数。

（5）Com_insert：插入操作的次数。

（6）Com_delete：删除操作的次数。

说明

MySQL 中存在查询 InnoDB 类型的表的一些参数。例如，Innodb_rows_read 参数表示 SELECT 语句查询的记录数；Innodb_rows_inserted 参数表示 INSERT 语句插入的记录数；Innodb_rows_updated 参数表示 UPDATE 语句更新的记录数；Innodb_rows_deleted 参数表示 DELETE 语句删除的记录数。

如果需要查询 MySQL 服务器的连接次数，可以执行下面的 SHOW STATUS 语句。

```
SHOW STATUS LIKE 'Connections';
```

通过这些参数可以分析 MySQL 数据库性能。然后根据分析结果，进行相应的性能优化。

例 19.1　使用 SHOW STATUS 语句实现查看 MySQL 服务器的连接和查询次数。查看 MySQL 服务器的连接和查询次数的语句执行效果如图 19.1 所示。**（实例位置：光盘\TM\sl\19\19.1）**

```
mysql> SHOW STATUS LIKE 'Connections';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| Connections   | 23    |
+---------------+-------+
1 row in set (0.00 sec)

mysql> SHOW STATUS LIKE 'Com_select';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| Com_select    | 6     |
+---------------+-------+
1 row in set (0.00 sec)

mysql> SHOW STATUS LIKE 'Slow_queries';
+---------------+-------+
| Variable_name | Value |
+---------------+-------+
| Slow_queries  | 0     |
+---------------+-------+
1 row in set (0.00 sec)

mysql> _
```

图 19.1　查看 MySQL 服务器的连接和查询次数

使用 SHOW STATUS 语句时，可以通过指定统计参数为 Connections、Com_select 和 Slow_queries，来实现显示 MySQL 服务器的连接数、查询次数和慢查询次数的功能，关键代码如下。

```
SHOW STATUS LIKE 'Connections';
SHOW STATUS LIKE 'Com_select';
SHOW STATUS LIKE 'Slow_queries';
```

19.2　优 化 查 询

查询是数据库最频繁的操作，提高查询速度可以有效地提高 MySQL 数据库的性能。本节将介绍优化查询的方法。

19.2.1　分析查询语句

分析查询语句在前面章节中都有应用，在 MySQL 中，可以使用 EXPLAIN 语句和 DESCRIBE 语句来分析查询语句。

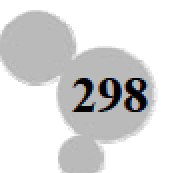
应用 EXPLAIN 关键字分析查询语句，其语法结构如下。

```
EXPLAIN   SELECT 语句;
```

"SELECT 语句"参数为一般数据库查询命令，如 SELECT * FROM students。

例 19.2　下面使用 EXPLAIN 语句分析一个查询语句，其代码如下。**（实例位置：光盘\TM\sl\19\19.2）**

```
EXPLAIN   SELECT *   FROM timeinfo ;
```



其运行结果如图 19.2 所示。

```
mysql> EXPLAIN  SELECT *  FROM timeinfo ;
+----+-------------+----------+------+---------------+------+---------+------+--
----+-------+
| id | select_type | table    | type | possible_keys | key  | key_len | ref  | r
ows | Extra |
+----+-------------+----------+------+---------------+------+---------+------+--
----+-------+
|  1 | SIMPLE      | timeinfo | ALL  | NULL          | NULL | NULL    | NULL |
  2 |       |
+----+-------------+----------+------+---------------+------+---------+------+--
----+-------+
1 row in set (0.02 sec)
```

图 19.2　应用 EXPLAIN 分析查询语句

其中各字段所代表的意义如下所示。

（1）id 列：指出在整个查询中 SELECT 的位置。

（2）table 列：存放所查询的表名。

（3）type 列：连接类型，该列中存储很多值，范围从 const 到 ALL。

（4）possible_keys 列：指出为了提高查找速度，在 MySQL 中可以使用的索引。

（5）key 列：指出实际使用的键。

（6）rows 列：指出 MySQL 需要在相应表中返回查询结果所检验的行数，为了得到该总行数，MySQL 必须扫描处理整个查询，再乘以每个表的行值。

（7）Extra 列：包含一些其他信息，设计 MySQL 如何处理查询。

在 MySQL 中，也可以应用 DESCRIBE 语句来分析查询语句。DESCRIBE 语句的使用方法与 EXPLAIN 语法是相同的，这两者的分析结果也大体相同。其中，DESCRIBE 的语法结构如下。

```
DESCRIBE SELECT 语句;
```

在命令提示符下输入如下命令。

```
describe select * from studentinfo;
```

其运行结果如图 19.3 所示。

```
mysql> describe select * from timeinfo;
+----+-------------+----------+------+---------------+------+---------+------+--
----+-------+
| id | select_type | table    | type | possible_keys | key  | key_len | ref  | r
ows | Extra |
+----+-------------+----------+------+---------------+------+---------+------+--
----+-------+
|  1 | SIMPLE      | timeinfo | ALL  | NULL          | NULL | NULL    | NULL |
  2 |       |
+----+-------------+----------+------+---------------+------+---------+------+--
----+-------+
1 row in set (0.00 sec)
```

图 19.3　应用 DESCRIBE 分析查询语句

将图 19.3 与图 19.2 对比，读者可以清楚地看出，其运行结果基本相同。分析查询也可以应用关键

字 DESCRIBE。



说明

DESCRIBE 可以缩写成 DESC。

19.2.2 索引对查询速度的影响

在查询过程中使用索引，势必会提高数据库查询效率，应用索引来查询数据库中的内容，可以减少查询的记录数，从而达到查询优化的目的。

例 19.3 通过对使用索引和不使用索引进行对比，来分析查询的优化情况。**（实例位置：光盘\TM\sl\19\19.3）**

首先，分析未使用索引时的查询情况，其代码如下。

```
explain select * from studentinfo where name= 'mrsoft ';
```

其运行结果如图 19.4 所示。

```
mysql> explain select * from studentinfo where name='mrsoft';
+----+-------------+-------------+------+---------------+------+---------+------
+------+-------------+
| id | select_type | table       | type | possible_keys | key  | key_len | ref
| rows | Extra       |
+----+-------------+-------------+------+---------------+------+---------+------
+------+-------------+
|  1 | SIMPLE      | studentinfo | ALL  | NULL          | NULL | NULL    | NULL
|    7 | Using where |
+----+-------------+-------------+------+---------------+------+---------+------
+------+-------------+
1 row in set (0.00 sec)
```

查询7条记录

图 19.4 未使用索引的查询情况

上述结果表明，表格字段 rows 下为 7，这意味着在执行查询过程中，数据库存在的 7 条数据都被查询了一遍，这样在数据存储量小的时候，查询不会有太大影响，试想当数据库中存储庞大的数据资料时，用户为了搜索一条数据而遍历整个数据库中的所有记录，将会耗费很多时间。现在，在 name 字段上建立一个名为 index_name 的索引。创建索引的代码如下。

```
CREATE INDEX index_name ON studentinfo(name);
```

上述代码的作用是在 studentinfo 表的 name 字段上添加索引。在建立索引完毕后，然后再应用关键字 EXPLAIN 分析执行情况，其代码如下所示。

```
explain select * from studentinfo where name = 'mrsoft ';
```

其运行结果如图 19.5 所示。



```
mysql> explain select * from studentinfo where name= 'mrsoft ';
+----+-------------+-------------+------+---------------+------------+---------+
-------+------+-------------+
| id | select_type | table       | type | possible_keys | key        | key_len |
 ref   | rows | Extra       |
+----+-------------+-------------+------+---------------+------------+---------+
-------+------+-------------+
|  1 | SIMPLE      | studentinfo | ref  | index_name    | index_name | 152     |
 const |    1 | Using where |
+----+-------------+-------------+------+---------------+------------+---------+
-------+------+-------------+
1 row in set (0.05 sec)
```

查询1条记录

图 19.5　使用索引后查询情况

从上述结果可以看出，由于创建的索引使访问的行数由 7 行减少到 1 行，所以，在查询操作中，使用索引不但会自动优化查询效率，同时也会降低服务器的开销。

19.2.3　使用索引查询

在 MySQL 中，索引可以提高查询的速度，但并不能充分发挥其作用，所以在应用索引查询时，也可以通过关键字或其他方式来对查询进行优化处理。

1．应用关键字 LIKE 优化索引查询

例 19.4　下面的示例应用关键字 LIKE，并且匹配字符串中含有百分号“%”符号，应用 EXPLAIN 语句执行如下命令。**（实例位置：光盘\TM\sl\19\19.4）**

```
EXPLAIN SELECT * FROM studentinfo WHERE name LIKE '%l';
```

其运行结果如图 19.6 所示。

```
mysql> explain select * from studentinfo where name like '%l';
+----+-------------+-------------+------+---------------+------+---------+------
+------+-------------+
| id | select_type | table       | type | possible_keys | key  | key_len | ref
| rows | Extra       |
+----+-------------+-------------+------+---------------+------+---------+------
+------+-------------+
|  1 | SIMPLE      | studentinfo | ALL  | NULL          | NULL | NULL    | NULL
|    7 | Using where |
+----+-------------+-------------+------+---------------+------+---------+------
+------+-------------+
1 row in set (0.22 sec)
```

图 19.6　应用关键字 LIKE 优化索引查询

从图 19.6 中可能看出其 rows 参数仍为“7”并没有起到优化作用，这是因为如果匹配字符串中，第一个字符为百分号“%”时，索引不会被使用，如果“%”所在匹配字符串中的位置不是第一位置，则索引会被正常使用，在命令提示符中输入如下命令。

```
EXPLAIN SELECT * FROM studentinfo WHERE name LIKE 'le%';
```

运行结果如图 19.7 所示。

```
mysql> explain select * from studentinfo where name like 'le%';
+----+-------------+-------------+-------+---------------+------------+---------
+------+------+-------------+
| id | select_type | table       | type  | possible_keys | key        | key_len
| ref  | rows | Extra       |
+----+-------------+-------------+-------+---------------+------------+---------
+------+------+-------------+
|  1 | SIMPLE      | studentinfo | range | index_name    | index_name | 152
| NULL |    1 | Using where |
+----+-------------+-------------+-------+---------------+------------+---------
+------+------+-------------+
1 row in set (0.00 sec)
```

图 19.7　正常应用索引的 LIKE 子句运行结果

2．查询语句中使用多列索引

多列索引在表的多个字段上创建一个索引。只有查询条件中使用了这些字段中的一个字段时，索引才会被正常使用。

应用多列索引在表的多个字段中创建一个索引，其命令如下。

```
CREATE INDEX index_student_info ON studentinfo(name,sex);
```



在应用 sex 字段时，索引不能被正常使用。这就意味着索引并未在 MySQL 优化中起到任何作用，故必须使用第一字段 name 时，索引才可以被正常使用，有兴趣的读者可以实际动手操作一下。这里不再赘述。

3．查询语句中使用关键字 OR

在 MySQL 中，查询语句只有包含关键字 OR 时，要求查询的两个字段必须同为索引，如果所搜索的条件中，有一个字段不为索引，则在查询中不会应用索引进行查询。其中，应用关键字 OR 查询索引的命令如下。

```
SELECT * FROM studentinfo WHERE name='Chris' or sex='M';
```

例 19.5　通过 EXPLAIN 来分析查询命令，在命令提示符中输入如下代码。**（实例位置：光盘\TM\sl\19\19.5）**

```
EXPLAIN SELECT * FROM studentinfo WHERE name='Chris' or sex='M';
```

其运行结果如图 19.8 所示。



```
mysql> explain select * from studentinfo where name='Chris' or sex='M';
+----+-------------+-------------+------+-----------------------------------+------+
---------+------+------+-------------+
| id | select_type | table       | type | possible_keys                     | key  |
 key_len | ref  | rows | Extra       |
+----+-------------+-------------+------+-----------------------------------+------+
---------+------+------+-------------+
|  1 | SIMPLE      | studentinfo | ALL  | index_name,index_student_info     | NULL |
 NULL    | NULL |    1 | Using where |
+----+-------------+-------------+------+-----------------------------------+------+
---------+------+------+-------------+
1 row in set (0.00 sec)
```

图 19.8　应用关键字 OR

从图 19.8 中可以看出，由于两个字段均为索引，故查询被优化。如果在子查询中存在没有被设置成索引的字段，则将该字段作为子查询条件时，查询速度不会被优化。

19.3　优化数据库结构

数据库结构是否合理，需要考虑是否存在冗余、对表的查询和更新的速度、表中字段的数据类型是否合理等多方面的内容。本节将介绍优化数据库结构的方法。

19.3.1　将字段很多的表分解成多个表

有些表在设计时设置了很多的字段。这些表中有些字段的使用频率很低。当这些表的数据量很大时，查询数据的速度就会很慢。本节将介绍优化这种表的方法。

对于这种字段特别多且有些字段的使用频率很低的表，可以将其分解成多个表。

例 19.6　下面的学生表中有很多字段，其中在 extra 字段中存储着学生的备注信息。有些备注信息的内容特别多，但是，备注信息很少使用。这样就可以分解出另外一个表，将这个表取名为 student_extra。表中存储两个字段，分别为 id 和 extra。其中，id 字段为学生的学号，extra 字段存储备注信息。student_extra 表的结构如表 19.9 所示。**（实例位置：光盘\TM\sl\19\19.6）**



```
mysql> desc student_extra;
+-------+---------+------+-----+---------+----------------+
| Field | Type    | Null | Key | Default | Extra          |
+-------+---------+------+-----+---------+----------------+
| id    | int(4)  | NO   | PRI | NULL    | auto_increment |
| extra | text    | YES  |     | NULL    |                |
+-------+---------+------+-----+---------+----------------+
2 rows in set (0.00 sec)

mysql> _
```

图 19.9　将字段很多的表分解成多个表

如果需要查询某个学生的备注信息，可以用学号（id）来查询。如果需要将学生的学籍信息与备

注信息同时显示时，可以将 student 表和 student_extra 表进行联表查询，查询语句如下。

```
SELECT * FROM student,student_extra WHERE student.id=student_extra.id;
```

通过这种分解，可以提高 student 表的查询效率。因此，遇到这种字段很多，而且有些字段使用不频繁的，可以通过这种分解的方式来优化数据库的性能。

19.3.2　增加中间表

有时需要经常查询某两个表中的几个字段。如果经常进行联表查询，会降低 MySQL 数据库的查询速度。对于这种情况，可以建立中间表来提高查询速度。本节将介绍增加中间表的方法。

先分析经常需要同时查询哪几个表中的哪些字段，然后将这些字段建立一个中间表，并将原来那几个表的数据插入到中间表中，之后就可以使用中间表来进行查询和统计。

例 19.7　创建中间表 temp_score 表保存经常要查询的学号、姓名和成绩等信息。**（实例位置：光盘\TM\sl\19\19.7）**

下面有个学生表 student 和分数表 score，这两个表的结构如图 19.10 所示。

```
mysql> desc student;
+----------+-------------+------+-----+---------+----------------+
| Field    | Type        | Null | Key | Default | Extra          |
+----------+-------------+------+-----+---------+----------------+
| id       | int(4)      | NO   | PRI | NULL    | auto_increment |
| name     | varchar(20) | NO   | MUL | NULL    |                |
| sex      | varchar(4)  | NO   |     | NULL    |                |
| age      | int(10)     | NO   |     | NULL    |                |
| zhuanye  | varchar(20) | NO   |     | NULL    |                |
| address  | varchar(30) | NO   |     | NULL    |                |
+----------+-------------+------+-----+---------+----------------+
6 rows in set (0.00 sec)

mysql> desc score;
+--------+-------------+------+-----+---------+----------------+
| Field  | Type        | Null | Key | Default | Extra          |
+--------+-------------+------+-----+---------+----------------+
| id     | int(4)      | NO   | PRI | NULL    | auto_increment |
| stu_id | int(4)      | NO   |     | NULL    |                |
| c_name | varchar(20) | YES  |     | NULL    |                |
| grade  | int(10)     | YES  |     | NULL    |                |
+--------+-------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)

mysql> _
```

图 19.10　增加中间表

实际中经常要查学生的学号、姓名和成绩。根据这种情况可以创建一个 temp_score 表。temp_score 表中存储 3 个字段，分别是 id、name 和 grade。CREATE 语句执行如下。

```
CREATE TABLE temp_score(id INT NOT NULL,
Name VARCHAR(20) NOT NULL,
grade FLOAT);
```

然后从 student 表和 score 表中将记录导入到 temp_score 表中。INSERT 语句如下。

```
INSERT INTO temp_score SELECT student.id,student.name,score.grade
FROM student,score WHERE student.id=score.stu_id;
```



将这些数据插入到 temp_score 表中以后，可以直接从 temp_score 表中查询学生的学号、姓名和成绩。这样就省去了每次查询时进行表连接的操作，可以提高数据库的查询速度。

19.3.3　优化插入记录的速度

插入记录时，索引、唯一性校验都会影响到插入记录的速度。而且，一次插入多条记录和多次插入记录所耗费的时间是不一样的。根据这些情况，分别进行不同的优化。本节将介绍优化插入记录的速度的方法。

1．禁用索引

插入记录时，MySQL 会根据表的索引对插入的记录进行排序。如果插入大量数据时，这些排序会降低插入记录的速度。为了解决这种情况，在插入记录之前应先禁用索引，等到记录都插入完毕后再开启索引。禁用索引的语句如下。

```
ALTER TABLE 表名 DISABLE KEYS;
```

重新开启索引的语句如下。

```
ALTER TABLE 表名 ENABLE KEYS;
```

对于新创建的表，可以先不创建索引。等到记录都导入以后再创建索引。这样可以提高导入数据的速度。

2．禁用唯一性检查

插入数据时，MySQL 会对插入的记录进行校验。这种校验也会降低插入记录的速度。可以在插入记录之前禁用唯一性检查。等到记录插入完毕后再开启。禁用唯一性检查的语句如下。

```
SET UNIQUE_CHECKS=0;
```

重新开启唯一性检查的语句如下。

```
SET UNIQUE_CHECKS=1;
```

3．优化 INSERT 语句

插入多条记录时，可以采取两种写 INSERT 语句的方式。第一种是一个 INSERT 语句插入多条记录。INSERT 语句的情形如下。

```
INSERT INTO food VALUES
(NULL,'果冻', 'CC 果冻厂',1.8, '2011', '北京'),
(NULL, '咖啡', 'CF 咖啡厂',25, '2012', '天津'),
(NULL, '奶糖', '旺仔奶糖',15, '2013', '广东');
```

第二种是一个 INSERT 语句只插入一条记录，执行多个 INSERT 语句来插入多条记录。INSERT 语句的情形如下。

```
INSERT INTO food VALUES(NULL, '果冻', 'CC 果冻厂',1.8, '2011', '北京');
```

```
INSERT INTO food VALUES(NULL, '咖啡', 'CF 咖啡厂',25, '2012', '天津');
INSERT INTO food VALUES(NULL, '奶糖', '旺仔奶糖',15, '2013', '广东');
```

第一种方式减少了与数据库之间的连接等操作，其速度比第二种方式要快。


说明

当插入大量数据时，建议使用一个 INSERT 语句插入多条记录的方式。而且，如果能用 LOAD DATA INFILE 语句，就尽量用 LOAD DATA INFILE 语句。因为 LOAD DATA INFILE 语句导入数据的速度比 INSERT 语句快。

19.3.4　分析表、检查表和优化表

分析表主要作用是分析关键字的分布。检查表主要作用是检查表是否存在错误。优化表主要作用是消除删除或者更新造成的空间浪费。本节将介绍分析表、检查表和优化表的方法。

1．分析表

MySQL 中使用 ANALYZE TABLE 语句来分析表，该语句的基本语法如下。

```
ANALYZE TABLE  表名 1[,表名 2…];
```

使用 ANALYZE TABLE 分析表的过程中，数据库系统会对表加一个只读锁。在分析期间，只能读取表中的记录，不能更新和插入记录。ANALYZE TABLE 语句能够分析 InnoDB 和 MyISAM 类型的表。

例 19.8　下面使用 ANALYZE TABLE 语句分析 score 表，分析结果如图 19.11 所示。（**实例位置：光盘\TM\sl\19\19.8**）

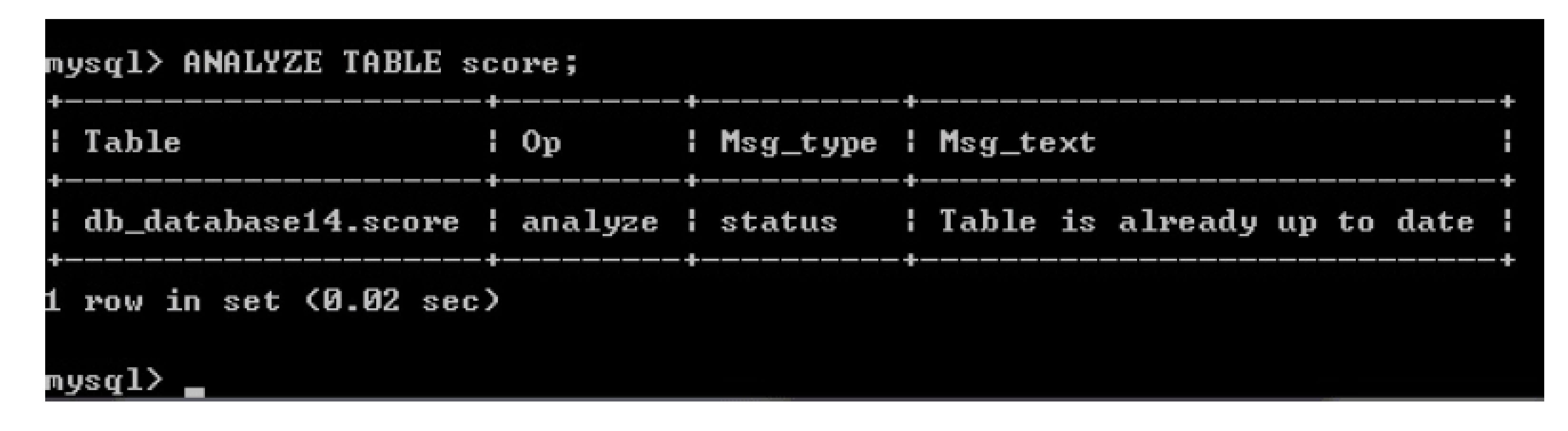


图 19.11　分析表

上面结果显示了 4 列信息，详细介绍如下。

（1）Table：表示表的名称。

（2）Op：表示执行的操作。analyze 表示进行分析操作，check 表示进行检查查找，optimize 表示进行优化操作。

（3）Msg_type：表示信息类型，其显示的值通常是状态、警告、错误或信息。

（4）Msg_text:：显示信息。

检查表和优化表之后也会出现这 4 列信息。



2．检查表

MySQL 中使用 CHECK TABLE 语句来检查表。CHECK TABLE 语句能够检查 InnoDB 和 MyISAM 类型的表是否存在错误。而且，该语句还可以检查视图是否存在错误。该语句的基本语法如下。

```
CHECK TABLE 表名 1[,表名 2…][option];
```

其中，option 参数有 5 个参数，分别是 QUICK、FAST、CHANGED、MEDIUM 和 EXTENDED。这 5 个参数的执行效率依次降低。option 选项只对 MyISAM 类型的表有效，对 InnoDB 类型的表无效。CHECK TABLE 语句在执行过程中也会给表加上只读锁。

3．优化表

MySQL 中使用 OPTIMIZE TABLE 语句来优化表。该语句对 InnoDB 和 MyISAM 类型的表都有效。但是，OPTILMIZE TABLE 与拒绝只能优化表中的 VARCHAR、BLOB 或 TEXT 类型的字段。OPTILMIZE TABLE 语句的基本语法如下。

```
OPTIMIZE TABLE 表名 1[,表名 2…];
```

通过 OPTIMIZE TABLE 语句可以消除删除和更新造成的磁盘碎片，从而减少空间的浪费。OPTIMIZE TABLE 语句在执行过程中也会给表加上只读锁。

说明

如果一个表使用了 TEXT 或者 BLOB 这样的数据类型，那么更新、删除等操作就会造成磁盘空间的浪费。因为，更新和删除操作后，以前分配的磁盘空间不会自动收回。使用 OPTIMIZE TABLE 语句就可以将这些磁盘碎片整理出来，以便以后再利用。

19.4　查询高速缓存

在 MySQL 中，用户通过 SELECT 语句查询数据时，该操作将结果集保存到一个特殊的高级缓存中，从而实现查询操作。首次查询后，当用户再次做相同查询操作时，MySQL 即可从高速缓存中检索结果。这样一来，既提高了查询速率，也起到优化查询的作用。

19.4.1　检验高速缓存是否开启

例 19.9　在 MySQL 中应用关键字 VARIABLES，以通配符形式查看服务器变量。其代码如下。**（实例位置：光盘\TM\sl\19\19.9）**

```
SHOW VARIABLES LIKE  '%query_cache %';
```

运行上述代码，其结果如图 19.12 所示。

```
1 row in set (0.05 sec)

mysql> show variables like '%query_cache%';
+------------------------------+---------+
| Variable_name                | Value   |
+------------------------------+---------+
| have_query_cache             | YES     |
| query_cache_limit            | 1048576 |
| query_cache_min_res_unit     | 4096    |
| query_cache_size             | 0       |
| query_cache_type             | ON      |
| query_cache_wlock_invalidate | OFF     |
+------------------------------+---------+
6 rows in set (0.05 sec)
```

图 19.12　检验高速缓存是否开启

下面对主要的参数进行说明。

（1）have_query_cache：表明服务器在默认安装条件下，是否已经配置查询高速缓存。

（2）query_cache_size：高速缓存分配空间，如果该空间为 86，则证明分配给高速缓存空间的大小为 86MB。如果该值为 0，则表明查询高速缓存已经关闭。

（3）query_cache_type：判断高速缓存开启状态，其变量值范围为 0~2。其中当该值为 0 或 OFF 时，表明查询高速缓存已经关闭；当该值为 1 或 ON 时表明高速缓存已经打开；其值为 2 或 DEMAND 时，表明要根据需要运行带有 SQL_CACHE 选项的 SELECT 语句，提供查询高速缓存。

19.4.2　使用高速缓存

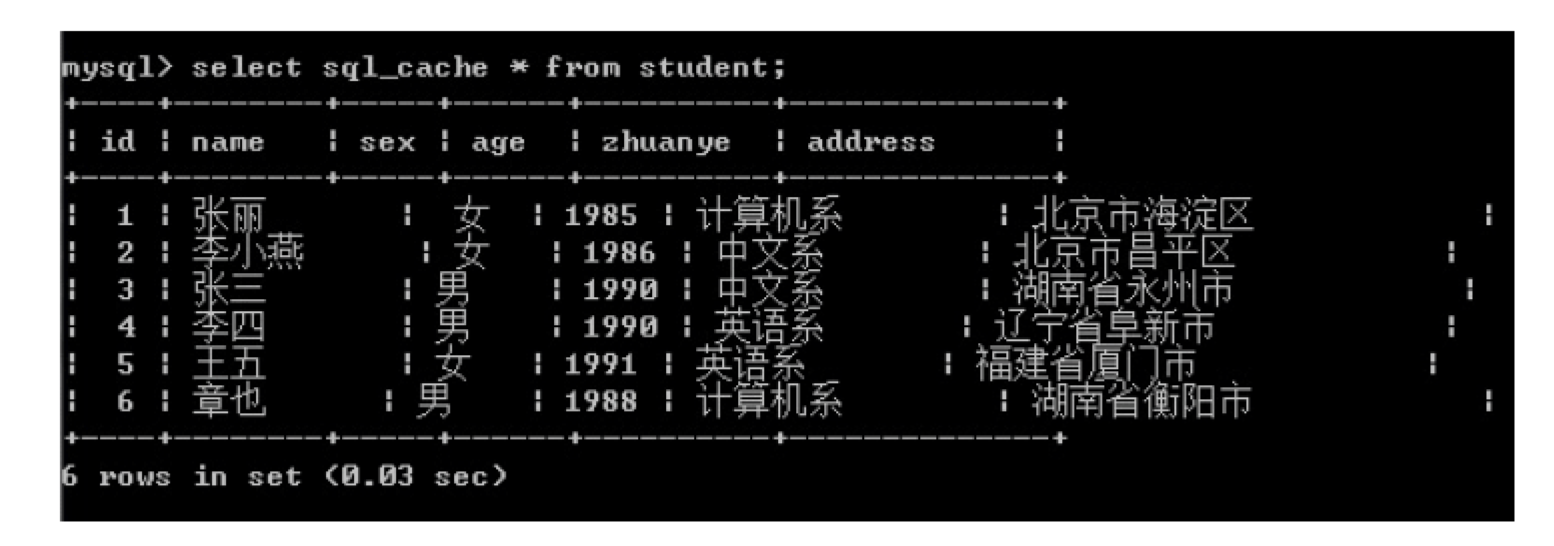
在 MySQL 中，查询高速缓存的具体语法结构如下。

```
SELECT SQL_CACHE * FROM 表名 ;
```

例 19.10　下面通过具体示例来查询高速缓存运行中的结果。在命令提示符下输入以下命令。**（实例位置：光盘\TM\sl\19\19.10）**

```
SELECT SQL_CACHE * FROM student ;
```

其运行结果如图 19.13 所示。



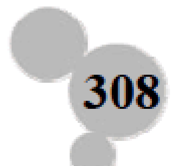
mysql> select sql_cache * from student;

id	name	sex	age	zhuanye	address
1	张丽	女	1985	计算机系	北京市海淀区
2	李小燕	女	1986	中文系	北京市昌平区
3	张三	男	1990	中文系	湖南省永州市
4	李四	男	1990	英语系	辽宁省阜新市
5	王五	女	1991	英语系	福建省厦门市
6	章也	男	1988	计算机系	湖南省衡阳市

6 rows in set (0.03 sec)

图 19.13　使用查询高速缓存运行结果

然后不使用高速缓存查询该数据表，其结果如图 19.14 所示。



```
mysql> select sql_no_cache * from student;
+----+--------+------+------+----------+--------------+
| id | name   | sex  | age  | zhuanye  | address      |
+----+--------+------+------+----------+--------------+
|  1 | 张丽   | 女   | 1985 | 计算机系 | 北京市海淀区 |
|  2 | 李小燕 | 女   | 1986 | 中文系   | 北京市昌平区 |
|  3 | 张三   | 男   | 1990 | 中文系   | 湖南省永州市 |
|  4 | 李四   | 男   | 1990 | 英语系   | 辽宁省阜新市 |
|  5 | 王五   | 女   | 1991 | 英语系   | 福建省厦门市 |
|  6 | 章也   | 男   | 1988 | 计算机系 | 湖南省衡阳市 |
+----+--------+------+------+----------+--------------+
6 rows in set (0.00 sec)
```

图 19.14　未使用查询高速缓存运行结果

如果经常运行查询高速缓存，将会提高 MySQL 数据库的性能。

说明

一旦表有变化，使用这个表的查询高速缓存将会失效，且将从高速缓存中删除。这样放置查询从旧表中返回无效数据。另外，不使用高速缓存查找可以应用 SQL_NO_CACHE 关键字。

19.5　优化多表查询

在 MySQL 中，用户可以通过连接来实现多表查询，在查询过程中，用户将表中的一个或多个共同字段进行连接，定义查询条件，返回统一的查询结果。这通常用来建立 RDBMS 常规表之间的关系。在多表查询中，可以应用子查询来优化多表查询，即在 SELECT 语句中嵌套其他 SELECT 语句。采用子查询优化多表查询的好处有很多，其中，可以将分步查询的结果整合成一个查询，这样就不需要再执行多个单独查询，从而提高了多表查询的效率。

例 19.11　下面通过一个实例来说明如何优化多表查询，首先在命令提示符下输入如下命令。**（实例位置：光盘\TM\sl\19\19.11）**

```
select address from student where id=(select id from student_extra wh
ere name='nihao');
```

其运行结果如图 19.15 所示。

```
mysql> select address from student where id=(select id from student_extra where
extra='nihao');
+--------------+
| address      |
+--------------+
| 北京市海淀区 |
+--------------+
1 row in set (0.03 sec)
```

图 19.15　应用一般 SELECT 嵌套子查询

下面应用优化算法，以便可以优化查询速度。在命令提示符下输入以下命令。

```
select address from student as stu,student_extra as stu_e where stu.id=stu_e.id and stu_e.extra='nihao';
```

以上命令的作用是将 student 和 student_extra 表分别设置别名 stu、stu_e，通过两个表的 id 字段建

立连接，并判断 student_extra 表中是否含有名称为“nihao”的内容，并将地址在屏幕上输出。该语句已经将算法进行优化，以便提高数据库的效率从而实现查询优化的效果。其运行结果如图 19.16 所示。

```
mysql> select address from student where id=(select id from student_extra where
extra='nihao');
+---------------+
| address       |
+---------------+
| 北京市海淀区        |
+---------------+
1 row in set (0.03 sec)
```

图 19.16　应用算法的优化查询

如果用户希望避免因出现 SELECT 嵌套而导致代码可读性下降，则用户可以通过服务器变量来进行优化处理，下面应用 SELECT 嵌套方式来查询数据，在命令提示符中输入如下命令。

```
select name from student where age> (select avg(age) from student_extra) ;
```

其运行结果如图 19.17 所示。

```
mysql> select name from student where age >(select avg(age) from student_extra);

+--------+
| name   |
+--------+
| 张丽      |
| 李小燕      |
| 张三      |
| 李四      |
| 王五      |
| 章也     |
+--------+
6 rows in set (0.01 sec)
```

图 19.17　应用 SELECT 嵌套查询数据

上述合并两个查询的速率将优于子查询运行速率，故采用服务器变量也可以优化查询。

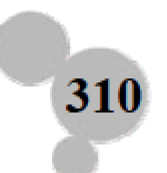
19.6　优化表设计

在 MySQL 数据库中，为了优化查询，使查询能够更加精炼、高效，在用户设计数据表的同时，也应该考虑一些因素。

首先，在设计数据表时应优先考虑使用特定字段长度，后考虑使用变长字段，如在用户创建数据表时，考虑创建某个字段类型为 varchar 而设置其字段长度为 255，但是在实际应用时，该用户所存储的数据根本达不到该字段所设置的最大长度，命令外如设置用户性别的字段，往往可以用“M”表示男性，“F”表示女性，如果给该字段设置长度为 varchar(50)，则该字段占用了过多列宽，这样不仅浪费资源，也会降低数据表的查询效率。适当调整列宽不仅可以减少磁盘空间，同时也可以使数据在进行处理时产生的 I/O 过程减少。将字段长度设置成其可能应用的最大范围可以充分地优化查询效率。

改善性能的另一项技术是使用 OPTIMIZE TABLE 命令处理用户经常操作的表。频繁地操作数据库中的特定表会导致磁盘碎片的增加，这样会降低 MySQL 的效率，故可以应用该命令处理经常操作的数据表，以便于优化访问查询效率。

在考虑改善表性能的同时，要检查用户已经建立的数据表，划分数据的优势在于可以使用户更好



地设计数据表，但是过多的表意味着性能降低，故用户应检查这些表。检查这些表是否有可能整合为一个表中，如没有必要整合，在查询过程中用户可以使用连接，如果连接的列采用相同的数据类型和长度，同样可以达到查询优化的作用。

说明

数据库表的类型 InnoDB 或 BDB 表处理行存储与 MyISAM 或 ISAM 表的情况不同。在 InnoDB 或 BDB 类型表中使用定长列，并不能提高其性能。

19.7　小　　结

本章对数据库优化的含义和查看数据性能参数的方法进行了详细讲解，然后介绍了优化查询的方法、优化数据库结构的方法和优化 MySQL 服务器的方法。优化查询的方法和优化数据库结果是本章的重点内容，优化查询部分主要介绍了索引对查询速度的影响。优化数据库结构部分主要介绍了如何对表进行优化。本章的难点是优化 MySQL 服务器，因为这部分涉及很多 MySQL 配置文件和配置文件中的参数。

19.8　实践与练习

1．实现在 MySQL 中使用 OPTIMIZE TABLE 语句来优化表。**（答案位置：光盘\TM\sl\19\19.12）**
2．使用 DESCRIBE 语句分析一个查询语句。**（答案位置：光盘\TM\sl\19\19.13）**

第20章

权限管理及安全控制

（视频讲解：10 分钟）

保护 MySQL 数据库的安全，就如同离开汽车时锁上车门，设置警报器。之所以这么做，主要是因为如果不采取这些基本但很有效的防范措施，那么汽车或者是车中的物品被盗的可能性会大大增加。本章将介绍有效保护 MySQL 数据库安全的一些有效措施。

通过阅读本章，读者可以：

- 了解安全保护策略
- 了解用各种命令实现对 MySQL 数据库的权限管理的方法
- 掌握使账户密码更安全的方法
- 掌握状态文件和日志文件

20.1　安全保护策略概述

要确保 MySQL 的安全，看看首先应当做点儿什么？

1．为操作系统和所安装的软件打补丁

如今打开计算机的时候，都会弹出软件的安全警告。虽然有些时候这些警告会给我们带来一些困扰，但是采取措施确保系统打上所有的补丁是绝对有必要的。利用攻击指令和 Internet 上丰富的工具，即使恶意用户在攻击方面没有多少经验，也可以毫无阻碍地攻击未打补丁的服务器。即使用户在使用托管服务器，也不要过分依赖服务提供商来完成必要的升级；相反，要坚持间隔性手动更新，以确保和补丁相关的事情都被处理妥当。

2．禁用所有不使用的系统服务

始终要注意在将服务器放入网络之前，已经消除所有不必要的潜在服务器攻击途径。这些攻击往往是不安全的系统服务带来的，通常运行在不为系统管理员所知的系统中。简言之，如果不打算使用一个服务，就禁用该服务。

3．关闭端口

虽然关闭未使用的系统服务是减少成功攻击可能性的好方法，不过还可以通过关闭未使用的端口来添加第二层安全。对于专用的数据库服务器，可以考虑关闭除 22（SSH 协议专用）、3306（MySQL 数据库使用的）和一些“工具”专用的（如 123（NTP 专用））等端口号在 1024 以下的端口。简言之，如果不希望在指定端口有数据通信，就关闭这个端口。除了在专用防火墙工具或路由器上做这些调整之外，还可以考虑利用操作系统的防火墙。

4．审计服务器的用户账户

特别是当已有的服务器再作为公司的数据库主机时，要确保禁用所有非特权用户，或者最好是全部删除。虽然 MySQL 用户和操作系统用户完全无关，但他们都要访问服务器环境，仅凭这一点就可能会有意地破坏数据库服务器及其内容。为完全确保在审计中不会有遗漏，可以考虑重新格式化所有相关的驱动器，并重新安装操作系统。

5．设置 MySQL 的 root 用户密码

对所有 MySQL 用户使用密码。客户端程序不需要知道运行它的人员的身份。对于客户/服务器应用程序，用户可以指定客户端程序的用户名。例如，如果 other_user 没有密码，任何人可以简单地用 mysql -u other_user db_name 冒充他人调用 mysql 程序进行连接。如果所有用户账户均存在密码，使用其他用户的账户进行连接将困难得多。

20.2　用户和权限管理

MySQL 数据库中的表与其他任何关系表没有区别，都可以通过典型的 SQL 命令修改其结构和数据。随着版本 3.22.11 的发行，可以使用 GRANT 和 REVOKE 命令。通过这些命令，可以创建和禁用用户，可以在线授予和撤回用户访问权限。由于语法严谨，这消除了由于不好的 SQL 查询（例如，忘记在 UPDATE 查询中加入 WHERE 字句）所带来的潜在危险的错误。

在 5.0 版本中，开发人员向 MySQL 管理工具又增加了两个新命令：CREATE USER 和 DROP USER。从而能更容易地增加新用户、删除和重命名用户，还增加了第三个命令 RENAME USER 用于重命名现有的用户。

20.2.1　使用 CREATE USER 命令创建用户

CREATE USER 用于创建新的 MySQL 账户。要使用 CREATE USER 语句，必须拥有 mysql 数据库的全局 CREATE USER 权限，或拥有 INSERT 权限。对于每个账户，CREATE USER 会在没有权限的 mysql.user 表中创建一个新记录。如果账户已经存在，则出现错误。使用自选的 IDENTIFIED BY 子句，可以为账户设置一个密码。user 值和密码的设置方法和 GRANT 语句一样。其命令的原型如下所示。

```
CREATE USER user [IDENTIFIED BY[PASSWORD 'PASSWORD']
[, user [IDENTIFIED BY[PASSWORD 'PASSWORD']]···
```

例 20.1　应用 CREATE USER 命令创建一个新用户，用户名为 mrsoft，密码为 mr，其运行结果如图 20.1 所示。**（实例位置：光盘\TM\sl\20\20.1）**

```
mysql> CREATE USER mrsoft IDENTIFIED BY 'mr';
Query OK, 0 rows affected (0.00 sec)
```

图 20.1　通过 CREATE USER 创建 mrsoft 的用户

20.2.2　使用 DROP USER 命令删除用户

如果存在一个或是多个账户被闲置，应当考虑将其删除，确保不会用于可能的违法的活动。利用 DROP USER 命令就能很容易地做到，它将从权限表中删除用户的所有信息，即来自所有授权表的账户权限记录。DROP USER 命令原型如下所示。

```
DROP USER user [, user] ···
```





DROP USER 不能自动关闭任何打开的用户对话。而且，如果用户有打开的对话，此时取消用户，则命令不会生效，直到用户对话被关闭后才生效。一旦对话被关闭，用户也被取消，此用户再次试图登录时将会失败。

例 20.2　应用 DROP USER 命令删除用户名为 mrsoft 的用户，其运行结果如图 20.2 所示。（**实例位置：光盘\TM\sl\20\20.2**）

```
mysql> DROP USER mrsoft;
Query OK, 0 rows affected (0.00 sec)
```

图 20.2　使用 DROP USER 删除 mrsoft 的用户

20.2.3　使用 RENAME USER 命令重命名用户

RENAME USER 语句用于对原有 MySQL 账户进行重命名。RENAME USER 语句的命令原型如下。

```
RENAME USER old_user TO new_user
[, old_user TO new_user] ···
```



如果旧账户不存在或者新账户已存在，则会出现错误。

例 20.3　应用 RENAME USER 命令将用户名为 mrsoft 的用户重新命名为 lh，其运行结果如图 20.3 所示。（**实例位置：光盘\TM\sl\20\20.3**）

```
mysql> RENAME USER mrsoft TO lh;
Query OK, 0 rows affected (0.00 sec)
```

图 20.3　使用 RENAME USER 对 mrsoft 的用户重命名

20.2.4　GRANT 和 REVOKE 命令

GRANT 和 REVOKE 命令用来管理访问权限，也可以用来创建和删除用户，但在 MySQL 5.0.2 中可以利用 CREATE USER 和 DROP USER 命令更容易地实现这些任务。GRANT 和 REVOKE 命令对于谁可以操作服务器及其内容的各个方面提供了多程度的控制，从谁可以关闭服务器，到谁可以修改特定表字段中的信息都能控制。表 20.1 中列出了使用这些命令可以授予或撤回的所有权限。

表 20.1　GRANT 和 REVOKE 管理权限

权　　限	意　　义
ALL [PRIVILEGES]	设置除 GRANT OPTION 之外的所有简单权限
ALTER	允许使用 ALTER TABLE
ALTER ROUTINE	更改或取消已存储的子程序
CREATE	允许使用 CREATE TABLE
CREATE ROUTINE	创建已存储的子程序
CREATE TEMPORARY TABLES	允许使用 CREATE TEMPORARY TABLE
CREATE USER	允许使用 CREATE USER、DROP USER、 RENAME USER 和 REVOKE ALL PRIVILEGES
CREATE VIEW	允许使用 CREATE VIEW
DELETE	允许使用 DELETE
DROP	允许使用 DROP TABLE
EXECUTE	允许用户运行已存储的子程序
FILE	允许使用 SELECT…INTO OUTFILE 和 LOAD DATA INFILE
INDEX	允许使用 CREATE INDEX 和 DROP INDEX
INSERT	允许使用 INSERT
LOCK TABLES	允许对拥有 SELECT 权限的表使用 LOCK TABLES
PROCESS	允许使用 SHOW FULL PROCESSLIST
REFERENCES	未被实施
RELOAD	允许使用 FLUSH
REPLICATION CLIENT	允许用户询问从属服务器或主服务器的地址
REPLICATION SLAVE	用于复制型从属服务器（从主服务器中读取二进制日志事件）
SELECT	允许使用 SELECT
SHOW DATABASES	SHOW DATABASES 显示所有数据库
SHOW VIEW	允许使用 SHOW CREATE VIEW
SHUTDOWN	允许使用 mysqladmin shutdown
SUPER	允许使用 CHANGE MASTER、KILL、PURGE MASTER LOGS 和 SET GLOBAL 语句，mysqladmin debug 命令；允许连接（一次），即使已达到 max_connections
UPDATE	允许使用 UPDATE
USAGE	“无权限”的同义词
GRANT OPTION	允许授予权限

如果授权表拥有含有 mixed-case 数据库或表名称的权限记录，并且 lower_case_table_names 系统变量已设置，则不能使用 REVOKE 撤销权限，必须直接操纵授权表（当 lower_case_table_names 已设置时，GRANT 将不会创建此类记录，但是此类记录可能已经在设置变量之前被创建了）。

授予的权限可以分为多个层级。

1. 全局层级

全局权限适用于一个给定服务器中的所有数据库，存储在 mysql.user 表中。GRANT ALL ON *.* 和 REVOKE ALL ON *.*只授予和撤销全局权限。

2. 数据库层级

数据库权限适用于一个给定数据库中的所有目标，存储在 mysql.db 和 mysql.host 表中。GRANT ALL ON db_name.*和 REVOKE ALL ON db_name.*只授予和撤销数据库权限。

3. 表层级

表权限适用于一个给定表中的所有列，存储在 mysql.tables_priv 表中。GRANT ALL ON db_name.tbl_name 和 REVOKE ALL ON db_name.tbl_name 只授予和撤销表权限。

4. 列层级

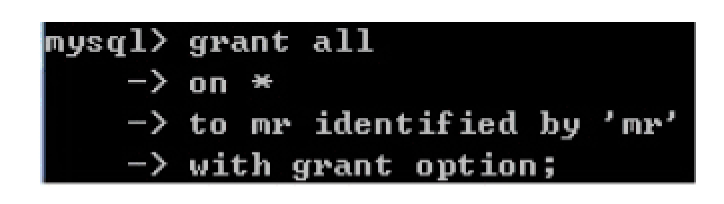
列权限适用于一个给定表中的单一列，这些权限存储在 mysql.columns_priv 表中。当使用 REVOKE 时，必须指定与被授权列相同的列。

5.子程序层级

CREATE ROUTINE、ALTER ROUTINE、EXECUTE 和 GRANT 权限适用于已存储的子程序。这些权限可以被授予为全局层级和数据库层级。而且，除了 CREATE ROUTINE 外，这些权限可以被授予为子程序层级，并存储在 mysql.procs_priv 表中。

例 20.4　下面创建一个管理员，以此来讲解 GRANT 和 REVOKE 命令的用法。创建一个管理员，可以输入如图 20.4 所示的命令。**（实例位置：光盘\TM\sl\20\20.4）**

以上命令授予用户名为 mr、密码为 mr 的用户使用所有数据库的所有权限，并允许他向其他人授予这些权限。如果不希望用户在系统中存在，可以按如图 20.5 所示的方式撤销。



```
mysql> grant all
    -> on *
    -> to mr identified by 'mr'
    -> with grant option;
```

图 20.4　创建管理员命令

```
mysql> revoke all privileges,grant
    -> from fred;
```

图 20.5　撤销用户命令

现在，按如图 20.6 所示的方式创建一个没有任何权限的常规用户。

```
mysql> grant ueage
    -> on books.*
    -> to mrsoft identified by 'magic123';
```

图 20.6　创建没有任何权限的常规用户

可以为用户 mrsoft 授予适当的权限，方式如图 20.7 所示。

```
mysql> grant select,insert,update,delete,index,alter,create,drop
    -> on books.*
    -> to mrsoft;
```

图 20.7　授予用户适当的权限命令



要完成对 mrsoft 用户授予权限，并不需要指定 mrsoft 的密码。

如果认为 mrsoft 权限过高，可以按如图 20.8 所示的方式减少一些权限。

当用户 mrsoft 不再需要使用数据库时，可以按如图 20.9 所示的方式撤销所有的权限。



图 20.8　减少权限的命令



图 20.9　撤销用户的所有权限



当用户使用 GRANT 和 REVOKE 命令更改用户权限后，退出 MySQL 系统，用户使用新账户名登录 MySQL 的时候，可能会因为没有刷新用户授权表而导致登录错误。这是因为在用户设置账号完毕后，只有重新加载授权表才能使之前设置的授权表生效。使用 FLUSH PRIVILEGES 命令可以重载授权表。该命令将在 20.3.1 节中讲解。

另外需要注意的是，只有如 root 这样拥有全部权限的用户才可以执行此命令。当用户重载授权表后，退出 MySQL 后，使用新创建的用户名即可正常登录 MySQL。

20.3　MySQL 数据库安全常见问题

20.3.1　权限更改何时生效

MySQL 服务器启动的时候以及使用 GRANT 和 REVOKE 语句的时候，服务器会自动读取 grant 表。但是，既然我们知道这些权限保存在什么地方以及它们是如何保存的，就可以手动修改它们。当手动更新它们的时候，MySQL 服务器将不会注意到它们已经被修改了。

我们必须向服务器指出已经对权限进行了修改，有 3 种方法可以实现这个任务。可以在 MySQL 命令提示符下（必须以管理员的身份登录进入）输入如下命令。

```
flush privileges;
```

这是更新权限最常使用的方法。或者，还可以在操作系统中运行：



```
mysqladmin flush-privileges
```

或者是

```
mysqladmin reload
```

此后，当用户下次再连接的时候，系统将检查全局级别权限；当下一个命令被执行时，将检查数据库级别的权限；而表级别和列级别权限将在用户下次请求的时候被检查。

20.3.2　设置账户密码

（1）可以用 mysqladmin 命令在 DOS 命令窗口中指定密码。

```
mysqladmin -u user_name -h host_name password "newpwd"
```

mysqladmin 命令重设服务器为 host_name，且用户名为 user_name 的用户的密码，新密码为“newpwd”。

（2）通过 set password 命令设置用户的密码。

```
set password for 'jeffrey'@'%' = password('biscuit');
```

只有以 root 用户（可以更新 mysql 数据库的用户）身份登录，才可以更改其他用户的密码。如果没有以匿名用户连接，省略 for 子句便可以更改自己的密码。

```
set password = password('biscuit');
```

（3）在全局级别下使用 GRANT USAGE 语句（在*.*）指定某个账户的密码，而不影响账户当前的权限。

```
GRANT USAGE ON *.* TO 'jeffrey'@'%' IDENTIFIED BY 'biscuit';
```

（4）在创建新账户时建立密码，要为 password 列提供一个具体值。

```
mysql -u root mysql
INSERT INTO user (Host,User,Password)
 -> VALUES('%','jeffrey',PASSWORD('biscuit'));
mysql> FLUSH PRIVILEGES;
```

（5）更改已有账户的密码，要应用 UPDATE 语句来设置 password 列值。

```
mysql -u root mysql
UPDATE user SET Password = PASSWORD('bagel')
        -> WHERE Host = '%' AND User = 'francis';
FLUSH PRIVILEGES;
```

说明

（1）当使用 SET PASSWORD、INSERT 或者 UPDATE 指定账户的密码时，必须用 PASSWORD()函数对它进行加密（唯一的特例是如果密码为空，则不需要使用 PASSWORD())。之所以使用 PASSWORD()函数是因为 user 表以加密方式保存密码，而不是明文。如果采用下面没有进行加密的方式设置密码，代码如下。

```
mysql -u root mysql
INSERT INTO user (Host,User,Password)
   -> VALUES('%','jeffrey','biscuit');
mysql> FLUSH PRIVILEGES;
```

结果是密码“biscuit”保存到 user 表后没有加密。当 jeffrey 使用该密码连接服务器时，其代码如下。

```
mysql -u jeffrey -pbiscuit test
Access denied
```

连接使用的密码值将被加密，并同保存在 user 表中的密码进行比较。但是，保存的值为字符串'biscuit'，因此比较将失败，服务器拒绝连接。

（2）如果使用 GRANT … IDENTIFIED BY 语句或 mysqladmin password 命令设置密码，它们均会自动加密密码。在这种情况下，不需要使用 PASSWORD()函数对密码进行加密。

20.3.3 使密码更安全

（1）在管理级别，切记不能将 mysql.user 表的访问权限授予任何非管理账户。

（2）采用下面的命令模式来连接服务器，以此来隐藏密码。命令如下。

```
mysql -u francis -p db_name
Enter password: ********
```

“*”字符指示输入密码的地方，输入的密码是不可见的。因为它对其他用户不可见，与在命令行上指定它相比，这样进入密码更安全。

（3）如果想要从非交互式方式下运行一个脚本调用一个客户端，就没有从终端输入密码的机会。其最安全的方法是让客户端程序提示输入密码或在适当保护的选项文件中指定密码。

20.4 状态文件和日志文件

MySQL 数据目录里还包含许多状态文件和日志文件，如表 20.2 所示。这些文件默认存放位置是相应的 MySQL 服务器的数据目录，其默认文件名是在服务器主机名上增加一些后缀而得到的。



表 20.2　MySQL 的状态文件和日志文件

文 件 类 型	默 认 名	文 件 内 容
进程 ID 文件	HOSTNAME.pid	MySQL 服务器进程的 ID
常规查询日志	HOSTNAME.log	连接/断开连接时间和查询信息
慢查询日志	HOSTNAME-slow.log	耗时很长的查询命令的文本
变更日志	HOSTNAME.nnn	创建/变更了数据表的结构定义或者修改了数据表内容的查询命令的文本
二进制变更日志	HOSTNAME-bin.nnn	创建/变更了数据表的结构定义或者修改了数据表内容的查询命令的二进制表示法
二进制变更日志的索引文件	HOSTNAME-bin.index	使用中的“二进制变更日志文件”的清单
错误日志	HOSTNAME.err	“启动/关机”事件和异常情况

20.4.1　进程 ID 文件

MySQL 服务器会在启动时把自己的进程 ID 写入 PID 文件，等运行结束时又会删除该文件。PID 文件是允许服务器本身被其他进程找到的工具。例如，如果运行 mysql.server，在系统关闭时，关闭 MySQL 服务器的脚本检查 PID 文件以决定它需要向哪个进程发出一个终止信号。

20.4.2　日志文件管理

默认情况下，所有日志创建于 mysqld 数据目录中。通过刷新日志，可以强制 mysqld 来关闭和重新打开日志文件（或者在某些情况下切换到一个新的日志）。当执行一个 FLUSH LOGS 语句或执行 mysqladmin flush-logs 或 mysqladmin refresh 时，出现日志刷新。如果正使用 MySQL 复制功能，从复制服务器将维护更多日志文件，被称为接替日志。日志文件的类型如表 20.3 所示。

表 20.3　日志文件的类型

日 志 文 件	记入文件中的信息类型
错误日志	记录启动、运行或停止 mysqld 时出现的问题
常规查询日志	记录建立的客户端连接和执行的语句
更新日志	记录更改数据的语句。不赞成使用该日志
二进制日志	记录所有更改数据的语句。还用于复制
慢查询日志	记录所有执行时间超过 long_query_time 秒的所有查询或不使用索引的查询

1．错误日志

错误日志记载着 MySQL 数据库系统的诊断和出错信息。如果 mysqld 莫名其妙地“死掉”并且 mysqld_safe 需要重新启动它，mysqld_safe 会在错误日志中写入一条 restarted mysqld 消息。如果 mysqld

注意到需要自动检查或者修复一个表，则错误日志中会写入一条消息。

在一些操作系统中，如果 mysqld“死掉”，错误日志将包含堆栈跟踪信息。跟踪信息可以用来确定 mysqld“死掉”的地方。可以用--log-error[=file_name]选项来指定 mysqld 保存错误日志文件的位置。如果没有指定 file_name 值，mysqld 使用错误日志名 host_name.err，并在数据目录中写入日志文件。如果执行 FLUSH LOGS，错误日志用-old 重新命名后缀，并且 mysqld 创建一个新的空日志文件（如果未给出--log-error 选项，则不会重新命名）。

如果不指定--log-error，或者（在 Windows 中）使用--console 选项，错误被写入标准错误输出 stderr。通常标准输出为服务器的终端。

在 Windows 中，如果未给出--console 选项，错误输出总是写入.err 文件。

2. 常规查询日志

如果想要知道 mysqld 内部发生了什么，应该用--log[=file_name]或-l [file_name]选项启动它。如果没有指定 file_name 的值，默认名是 host_name.log。所有连接和语句被记录到日志文件。如果怀疑在客户端发生了错误并想确切地知道该客户端发送给 mysqld 的语句时，该日志可能非常有用。

mysqld 按照它接收的顺序记录语句到查询日志，这可能与执行的顺序不同。与更新日志和二进制日志不同，它们在查询执行后，任何一个锁释放前记录日志（查询日志还包含所有语句，而二进制日志不包含只查询数据的语句）。

服务器重新启动和日志刷新不会产生一般的新查询日志文件（尽管刷新关闭并重新打开一般查询日志文件）。在 UNIX 操作系统中，可以通过下面的命令重新命名文件并创建一个新文件。

```
shell> mv hostname.log hostname-old.log
shell> mysqladmin flush-logs
shell> cp hostname-old.log to-backup-directory
shell> rm hostname-old.log
```

在 Windows 中，服务器打开日志文件期间不能重新命名日志文件。首先，必须停止服务器；然后重新命名日志文件；最后，重启服务器来创建新的日志文件。

3. 二进制日志

二进制日志包含所有更新的数据或者已经潜在更新的数据（例如，没有匹配任何行的一个 DELETE）的所有语句。语句以“事件”的形式保存，它描述数据更改。

说明

> 二进制日志已经代替了老的更新日志，更新日志在 MySQL 5.1 中不再使用。

二进制日志还包含关于每个更新数据库的语句的执行时间信息，但不包含没有修改任何数据的语句。如果想要记录所有语句（例如，为了识别有问题的查询），应使用一般查询日志。

二进制日志的主要目的是在恢复时能够最大可能地更新数据库，因为二进制日志包含备份后进行的所有更新。

二进制日志还用于在主服务器上记录所有将发送给从服务器的语句。



当用--log-bin[=file_name]选项启动时，mysqld 写入包含所有更新数据的 SQL 命令的日志文件。如果未给出 file_name 值，默认名为-bin 后面所跟的主机名。如果给出了文件名，但没有包含路径，则文件被写入数据目录。如果在日志名中提供了扩展名（例如，--log-bin=file_name.extension），则扩展名会被忽略。

mysqld 在每个二进制日志名后面添加一个数字扩展名。每次启动服务器或刷新日志时该数字则增加。如果当前的日志大小达到 max_binlog_size，还会自动创建新的二进制日志。如果正在使用大的事务，二进制日志超过 max_binlog_size，事务全写入一个二进制日志中，绝对不要写入不同的二进制日志中。为了能够使当前用户知道还使用哪个不同的二进制日志文件，mysqld 还创建一个二进制日志索引文件，包含所有使用的二进制日志文件的文件名。默认情况下与二进制日志文件的文件名相同，扩展名为.index。可以用--log-bin-index[=file_name]选项更改二进制日志索引文件的文件名。当 mysqld 在运行时，不应手动编辑该文件；如果这样做将会使 mysqld 变得混乱。

可以用 RESET MASTER 语句删除所有二进制日志文件，或用 PURGE MASTER LOGS 只删除部分二进制文件。

如果系统正进行二进制文件复制，应确保没有从服务器在使用旧的二进制日志文件，方可删除它们。一种方法是每天执行一次 mysqladmin flush-logs 并删除三天前的所有日志。可以手动删除，或最好使用 PURGE MASTER LOGS，该语句还会安全地更新二进制日志索引文件（可以采用日期参数）。

具有 SUPER 权限的客户端可以通过 SET SQL_LOG_BIN=0 语句禁止将自己的语句记入二进制记录。可以用 mysqlbinlog 实用工具检查二进制日志文件。

如果想要重新处理日志的语句，这很有用。例如，可以从二进制日志更新 MySQL 服务器，方法如下。

```
shell> mysqlbinlog log-file | mysql -h server_name
```

如果用户正使用事务，必须使用 MySQL 二进制日志进行备份，而不能使用旧的更新日志。

查询结束后、锁定被释放前或提交完成后的事务，则立即将数据记入二进制日志，这样可以确保按执行顺序记入日志。

对非事务表的更新执行完毕后立即保存到二进制日志中。对于事务表，如 BDB 或 InnoDB 表，所有更改表的更新（UPDATE、DELETE 或 INSERT）被存入缓存中，直到服务器接收到 COMMIT 语句。在该点，当用户执行完 COMMIT 之前，mysqld 将整个事务写入二进制日志。当处理事务的线程启动时，它为缓冲查询分配 binlog_cache_size 大小的内存。如果语句大于该值，线程则打开临时文件来保存事务。线程结束后临时文件被删除。

binlog_cache_use 状态变量显示使用该缓冲区（也可能是临时文件）保存语句的事务数量。binlog_cache_disk_use 状态变量显示这些事务中实际上有多少必须使用临时文件。这两个变量可以用于将 binlog_cache_size 调节到足够大的值，以避免使用临时文件。

max_binlog_cache_size（默认 4GB）可以用来限制用来缓存多语句事务的缓冲区总大小。如果某个事务大于该值，将会失败并执行回滚操作。

如果正使用更新日志或二进制日志，当使用 CREATE … SELECT or INSERT … SELECT 时，并行插入被转换为普通插入。这样通过在备份时使用日志可以确保重新创建表的备份。

默认情况下，并不是每次写入时都将二进制日志与硬盘同步。因此如果操作系统或机器（不仅是

MySQL 服务器）崩溃，有可能二进制日志中最后的语句丢失了。要想防止这种情况，可以使用 sync_binlog 全局变量（1 是最安全的值，但也是最慢的），使二进制日志在每 N 次二进制日志写入后与硬盘同步。即使 sync_binlog 设置为 1，出现崩溃时，也有可能表内容和二进制日志内容之间存在不一致性。例如，如果使用 InnoDB 表，MySQL 服务器处理 COMMIT 语句，它将整个事务写入二进制日志并将事务提交到 InnoDB 中。如果在两次操作之间出现崩溃，重启时，事务被 InnoDB 回滚，但仍然存在二进制日志中。可以用--innodb-safe-binlog 选项解决该问题，可以增加 InnoDB 表内容和二进制日志之间的一致性。



在 MySQL 5.1 中不需要--innodb-safe-binlog，由于引入了 XA 事务支持，该选项作废了。

该选项可以提供更大程度的安全，还应对 MySQL 服务器进行配置，使每个事务的二进制日志（sync_binlog =1）和（默认情况为真）InnoDB 日志与硬盘同步。该选项的效果是崩溃后重启时，在滚回事务后，MySQL 服务器从二进制日志剪切回滚的 InnoDB 事务。这样可以确保二进制日志反馈 InnoDB 表的确切数据等，并使从服务器与主服务器保持同步（不接收回滚的语句）。

注意，即使 MySQL 服务器更新其他存储引擎而不是 InnoDB，也可以使用--innodb-safe-binlog。在 InnoDB 崩溃恢复时，只从二进制日志中删除影响 InnoDB 表的语句/事务。如果崩溃恢复时 MySQL 服务器发现二进制日志变短了（即至少缺少一个成功提交的 InnoDB 事务），如果 sync_binlog =1 并且硬盘/文件系统的确能根据需要进行同步（有些不需要）则不会发生，则输出错误消息 （"二进制日志<名>比期望的要小"）。在这种情况下，二进制日志不准确，复制应从主服务器的数据快照开始。

4．慢查询日志

慢查询日志记载着执行用时较长的查询命令，这里所说的“长”是由 MySQL 服务器变量 long_query_time（以秒为单位）定义的。每出现一个慢查询，MySQL 服务器就会给它的 slow_queries 状态计算器加上一个 1。

用--log-slow-queries[=file_name]选项启动时，mysqld 写一个包含所有执行时间超过 long_query_time 秒的 SQL 语句的日志文件。

如果没有给出 file_name 值， 默认为主机名，后缀为-slow.log。如果给出了文件名，但不是绝对路径名，文件则写入数据目录。

语句执行完并且所有锁释放后记入慢查询日志。记录顺序可以与执行顺序不相同。

慢查询日志可以用来找到执行时间长的查询，可以用于优化。但是，检查又长又慢的查询日志会很困难。要想容易些，可以使用 mysqldumpslow 命令获得日志中显示的查询摘要来处理慢查询日志。

在 MySQL 5.1 的慢查询日志中，不使用索引的慢查询同使用索引的查询一样记录。要想防止不使用索引的慢查询记入慢查询日志，使用--log-short-format 选项。

在 MySQL 5.1 中，通过--log-slow-admin-statements 服务器选项，可以请求将慢管理语句，例如 OPTIMIZE TABLE、ANALYZE TABLE 和 ALTER TABLE 写入慢查询日志。

用查询缓存处理的查询不加到慢查询日志中，因为表有零行或一行而不能从索引中受益的查询也不写入慢查询日志。

5．日志文件维护

MySQL 服务器可以创建各种不同的日志文件，从而可以很容易地看见所进行的操作。但是，必须定期清理这些文件，确保日志不会占用太多的硬盘空间。

当启用日志使用 MySQL 时，可能想要不时地备份并删除旧的日志文件，并告诉 MySQL 开始记入新文件。在 Linux （Red Hat）的安装上，可为此使用 mysql-log-rotate 脚本。如果从 RPM 分发安装 MySQL，脚本应该自动被安装了。

在其他系统上，必须自己安装短脚本，可从 cron 等入手处理日志文件。可以通过 mysqladmin flush-logs 或 SQL 语句 FLUSH LOGS 来强制 MySQL 开始使用新的日志文件。

日志清空执行的操作如下。

（1）如果使用标准日志（--log）或慢查询日志（--log-slow-queries），关闭并重新打开日志文件。（默认为 mysql.log 和`hostname`-slow.log）。

（2）如果使用更新日志（--log-update）或二进制日志（--log-bin），关闭日志并且打开有更高序列号的新日志文件。

如果只使用更新日志，只需要重新命名日志文件，然后在备份前清空日志。例如：

```
shell> cd mysql-data-directory
shell> mv mysql.log mysql.old
shell> mysqladmin flush-logs
```

然后做备份并删除 mysql.old。

6．日志失效处理

激活日志功能的弊病之一是随着日志的增加而产生的大量信息，生成的日志文件有可能会填满整个磁盘。如果 MySQL 服务器非常繁忙且需要处理大量的查询。用户既想保持有足够的空间来记录 MySQL 服务器的工作情况日志，又想防止日志文件无限制地增长，就需要应用一些日志文件的失效处理技术。进行日志失效处理的方式主要有以下几种。

1）日志轮转

该方法适用于常规查询日志和慢查询日志这些文件名固定的日志文件，在日志轮转时，应进行日志刷新操作（mysqladmin flush-logs 命令或 flush logs 语句），以确保缓存在内存中的日志信息写入磁盘。

日志轮转的操作过程是这样的（假设日志文件的名字是 log）：首先，第一次轮转时，把 log 更名为 log.1，然后服务器再创建一个新的 log 文件；在第二次轮转时，再把 log.1 更名为 log.2，把 log 更名为 log.1，然后服务器再创建一个新的 log 文件；如此循环，创建一系列的日志文件。当到达日志轮转失效位置时，下次轮转就不再对它进行更名，直接把最后一个日志文件覆盖掉。例如，如果每天进行一次日志轮转并想保留最后 7 天的日志文件，就需要保留 log.1~log.7 共 7 个日志文件，等下次轮转时，用 log.6 覆盖原来的 log.7 成新的 log.7，原来的 log.7 就自然失效。

日志轮转的频率和需要保留的老日志时间取决于 MySQL 服务器的繁忙程度（服务器越繁忙，生成的日志信息就越多）和用户分配用于存放老日志的磁盘空间。

UNIX 系统允许对 MySQL 服务器已经打开并正在使用的当前日志文件进行更名，日志刷新操作将

关闭当前日志文件并打开一个新日志文件，用原来的名字创建一个新的日志文件。文件名固定不变的日志文件可以用下面这个 shell 脚本来进行轮转。

```
#!/bin/sh
# rotate_fixed_logs.sh - rotate MySQL log file that has a fixed name
# Argument 1:log file name
if [ $# -ne 1 ]; then
    echo "Usage: $0 logname" 1>&2
    exit 1
if
logfile=$1
mv $logfile.6 $logfile.7
mv $logfile.5 $logfile.6
mv $logfile.4 $logfile.5
mv $logfile.3 $logfile.4
mv $logfile.2 $logfile.3
mv $logfile.1 $logfile.2
mv $logfile $logfile.1
mysqladmin flush-logs
```

这个脚本以日志文件名作为参数，既可以直接给出日志文件的完整路径名，也可以先进入日志文件所在的目录再给出日志文件的文件名。比如说，如果想对/usr/mysql/data 目录名为 log 的日志进行轮转，可以使用下面这条命令。

```
% rotate_fixed_logs.sh /usr/mysql/data/log
```

也可以使用下面的命令。

```
% cd/usr/mysql/data
% rotate_fixed_logs.sh log
```

为确保管理员自己总是存在权限对日志文件进行更名，最好是在以 mysqladm 为登录名上机时运行这个脚本，这里需要注意的是，在这个脚本里的 mysqladmin 命令行上没有给出-u 或-p 之类的连接选项参数。

如果用户已经把执行 mysql 客户程序时要用到的连接参数保存到了 mysqladmin 程序的 my.cnf 选项文件里，就不用在这个脚本中的 mysqladmin 命令行上再次给出它们。

如果用户没有使用选项文件，就必须使用-u 和-p 选项告诉 mysqladmin 使用哪个 MySQL 账户（这个 MySQL 账户必须具备日志刷新操作所需要的权限）去连接 MySQL 服务器。这样，MySQL 账户的口令将会出现在 rotate_fixed_logs.sh 脚本的代码里，所以为了防止这个脚本成为一个安全漏洞，这里建议读者专门创建一个除了能对日志进行刷新以外没有其他任何权限的 MySQL 账户（即一个具备且仅具备 RELOAD 权限的 MySQL 账户），将该账户的口令写到脚本代码里，最后再将这个脚本设置成只允许 mysqladm 用户去编辑和使用。下面这条 GRANT 语句将以 mrsoft 为用户名、以 mrsoftpass 为口令创建出一个如上所述的 MySQL 账户来。

```
GRANT RELOAD ON *.* TO 'flush'@'localhost' IDENTIFIED BY 'mrsoftpass';
```

创建出这个账户之后，再把 rotate_fixed_logs.sh 脚本中的 mysqladmin 命令行改写为如下所示的



命令。

```
mysqladmin –u mrsoft –pmrsoftpass mrsoft-logs
```

在 Linux 系统上的 MySQL 发行版本中带有一个用来安装 mysql-log-rotate 日志轮转脚本的 logrotate 工具，所以不必非得使用 rotate_fixed_logs.sh 或者自行编写其他的类似脚本。如用 RPM 安装，则在/usr/share/mysql 目录；如用二进制方式安装，则在 MySQL 安装目录的 support-files 目录；如用源码安装，则在安装目录的 share/mysql 目录中。

Windows 系统上的日志轮转与 UNIX 系统的不太一样。如果试图对一个已经被 MySQL 服务器打开并使用着的日志文件进行更名操作，就会发生“file in use”（文件已被打开）错误。要在 Windows 系统上对日志进行轮转，就得先停止 MySQL 服务器，然后对文件进行更名，最后再重新启动 MySQL 服务器，在 Windows 系统上启动和停止 MySQL 服务器的步骤前面已经介绍了。下面是一个进行日志更名的批处理文件。

```
@echo off
REM rotate_fixed_logs.bat – rotate MySQL log file that has a fixed name
if not "%1" == "" goto ROTATE
    @echo Usage: rotate_fixed_logs logname
    goto DONE
:ROTATE
set logfile=%1
erase %logfile%.7
rename %logfile%.6 %logfile%.7
rename %logfile%.5 %logfile%.6
rename %logfile%.4 %logfile%.5
rename %logfile%.3 %logfile%.4
rename %logfile%.2 %logfile%.3
rename %logfile%.1 %logfile%.2
rename %logfile% %logfile%.1
:DONE
```

这个批处理程序的用法与 rotate_fixed_logs.sh 脚本差不多，它也需要提供一个将被轮转的日志文件名作为参数，如下所示。

```
c:\>rotate_log c:\mysql\data\log
```

或者如下所示。

```
c:\>cd\mysql\data
c:\> rotate_fixed_logs log
```

说明

在最初几次执行日志轮转脚本的时候，日志文件的数量尚未达到预设的上限值，脚本会提示找不到某几个文件，这是正常的。

2）以时间为依据对日志进行失效处理

该方法将定期删除超过指定时间的日志文件，适用于变更日志和二进制日志等文件名用数字编号标识的日志文件。

下面是一个用来对以数字编号作为扩展名的日志文件进行失效处理的脚本。

```
#!/usr/bin/perl -w
# expire_numbered_logs.pl – look through a set of numbered MySQL
# log files and delete those that are more than a week old.
# Usage: expire_numbered_logs.pl   logfile ...
use strict;
die "Usage: $0 logfile ...\n" if @ARGV == 0;
my $max_allowed_age = 7;         #max allowed age in days
foreach my $file (@ARGV)         #check each argument
{
    unlink ($file) if -e $file && -M $file >= $max_allowed_age;
}
exit(0);
```

以上这个脚本是用 Perl 语言写的。Perl 是一种跨平台的脚本语言，用它编写出来的脚本在 UNIX 和 Windows 系统上皆可使用。这个脚本也需要提供一个被轮转的日志文件名作为参数，下面是在 UNIX 系统上的用法。

```
% expire_numbered_logs.pl /usr/mysql/data/update.[0-9]*
```

或者是

```
% cd/usr/mysql/data
% expire_numbered_logs.pl update.[0-9]*
```

注意

如果传递给这个脚本的文件名参数不正确，就会很危险。例如，将“*”作为这个脚本的文件名参数，即

```
% cd/usr/mysql/data
% expire_numbered_logs.pl *
```

这样就会把/usr/mysql/data 目录里更新时间大于 7 天的所有文件（不仅仅是日志文件）全都删除。由于通过 DatePicker 对象获取到的月份从 0 ~ 11，而不是月份中的 1 ~ 12，所以需要将获取到的结果再加 1，才能代表真正的月份。

3）镜像机制

将日志文件镜像到所有的从服务器上，就需要使用镜像机制，用户必须知道主服务器有多少个从服务器，哪些正在运行，并需依次连接每一个从服务器，同时发出 show slave status 语句以确定它正处理主服务器的哪个二进制日志文件（语句输出列表的 Master_Log_File 项），只有所有的从服务器都不会用到的日志文件才能删除。例如，本地 MySQL 服务器是主服务器，它有两个从 MySQL 服务器 S1 和 S2。在主服务器上有 5 个二进制日志文件，它们的名字是 mrlog0.38~mrlog0.42。

SHOW SLAVE STATUS 语句在 S1 上的执行结果如下。



```
mysql> SHOW SLAVE STATUS\G
…
Master_Log_File:mrlog.41
…
```

在 S2 上的执行结果如下。

```
mysql> SHOW SLAVE STATUS\G
…
Master_Log_File:mrlog.40
…
```

这样，我们就知道从服务器仍在使用的、最低编号的二进制日志是 mrlog.40，而编号比它更小的那些二进制日志，因为不再有从服务器需要用到它们，所以已经可以安全地删掉。于是，连接到主服务器并发出下面的语句：

```
mysql> PURGE MASTER LOGS TO 'mrlog.040';
```

在主服务器上发出的这条命令将把编号小于 40 的二进制日志文件删除。

20.5　小　　结

本章对 MySQL 数据库的账户管理和权限管理的内容进行了详细讲解，其中，账户管理和权限管理是本章的重点内容。这两部分中的密码管理、授权和收回权限是重中之重，因为这些内容涉及 MySQL 数据库的安全。希望读者能够认真学习这部分的内容。

20.6　实践与练习

1．实现创建一个名称为 mr 的用户，然后再将其删除。**（答案位置：光盘\TM\sl\20\20.5）**

2．使用 set password 命令将刚刚创建的 mr 用户的密码设置为 111。**（答案位置：光盘\TM\sl\20\20.6）**

第21章

PHP 管理 MySQL 数据库中的数据

（视频讲解：22 分钟）

PHP 是一种非常适合编写动态 Web 网页的脚本语言，用它编写出来的代码能够方便地嵌入到 Web 页面里。当这个 Web 页面被访问时，嵌入在其中的 PHP 代码就会被执行并生成动态的 HTML 内容，而这些内容将作为 Web 页面的一部分被送往用户的 Web 浏览器去显示。

通过阅读本章，读者可以：

- 了解 PHP 操作 MySQL 数据库的步骤
- 掌握通过 PHP 函数操作 MySQL 数据库的方法
- 掌握使用 PHP 操作 MySQL 数据库中的数据的方法
- 掌握 PHP 操作 MySQL 事务的方法
- 掌握在 PHP 中操作 MySQL 的存储过程的方法
- 了解 PHP 操作 MySQL 数据库出现的常见问题及解决方法

21.1　PHP 语言概述

21.1.1　PHP 的概念

PHP 是 Hypertext Preprocessor（超文本预处理器）的缩写，是一种服务器端、跨平台、HTML 嵌入式的脚本语言。其独特的语法混合了 C 语言、Java 语言和 Perl 语言的特点，是一种被广泛应用的开源式的多用途脚本语言，尤其适合 Web 开发。

21.1.2　PHP 的特点

PHP 起源于 1995 年，由 Rasmus Lerdorf 开发。目前已有超过 2 200 万个网站、1.5 万家公司、450 万程序开发人员在使用 PHP 语言，它是目前动态网页开发中使用最为广泛的语言之一。PHP 是生于网络、用于网络、发展于网络的一门语言，它一诞生就被打上了自由发展的烙印。目前在国内外有数以千计的个人和组织的网站在以各种形式和各种语言学习、发展和完善它，并不断地公布最新的应用和研究成果。PHP 能运行在包括 Windows、Linux 等在内的绝大多数操作系统环境中，常与免费 Web 服务器软件 Apache 和免费数据库 MySQL 配合使用于 Linux 平台上，具有最高的性价比，这 3 种技术的结合号称“黄金组合”。下面介绍 PHP 开发语言的特点。

1. 速度快

PHP 是一种强大的 CGI 脚本语言，语法混合了 C、Java、Perl 和 PHP 式的新语法，执行网页速度比 CGI、Perl 和 ASP 更快，而且内嵌 Zend 加速引擎，性能稳定快速。这是它的第一个突出的特点。

2. 支持面向对象

面向对象编程（OOP）是当前的软件开发趋势，PHP 对 OOP 提供了良好的支持。可以使用 OOP 的思想来进行 PHP 的高级编程，对于提高 PHP 编程能力和规划好 Web 开发构架都非常有意义。

3. 实用性

由于 PHP 是一种面向对象的、完全跨平台的新型 Web 开发语言，所以无论从开发者角度考虑还是从经济角度考虑，都是非常实用的。PHP 语法结构简单，易于入门，很多功能只需一个函数就可以实现，并且很多机构都相继推出了用于开发 PHP 的 IDE 工具。

4. 功能强大

PHP 在 Web 项目开发过程中具有极其强大的功能，而且实现相对简单，主要表现在如下几点。

（1）可操纵多种主流与非主流的数据库，如 MySQL、Access、SQL Server、Oracle、DB2 等，其中，PHP 与 MySQL 是现在绝佳的组合，可以跨平台运行。

（2）可与轻量级目录访问协议进行信息交换。

（3）可与多种协议进行通信，包括 IMAP、POP3、SMTP、SOAP 和 DNS 等。

（4）使用基于 POSIX 和 Perl 的正则表达式库解析复杂字符串。

（5）可以实现对 XML 文档进行有效管理及创建和调用 Web 服务等操作。

5. 可选择性

PHP 可以采用面向过程和面向对象两种开发模式，并向下兼容，开发人员可以从所开发网站的规模和日后维护等多角度考虑，以选择所开发网站应采取的模式。

PHP 进行 Web 开发过程中使用最多的是 MySQL 数据库。PHP 5.0 以上版本中不仅提供了早期 MySQL 数据库操纵函数，而且提供了 MySQLi 扩展技术对 MySQL 数据库的操纵，这样开发人员可以从稳定性和执行效率等方面考虑操纵 MySQL 数据库的方式。

6. 成本低

PHP 具有很好的开放性和可扩展性，属于自由软件，其源代码完全公开，任何程序员为 PHP 扩展附加功能非常容易。在很多网站上都可以下载到最新版本的 PHP。目前，PHP 主要是基于 Web 服务器运行的，支持 PHP 脚本运行的服务器有多种，其中最有代表性的为 Apache 和 IIS。PHP 不受平台束缚，可以在 UNIX、Linux 等众多版本的操作系统中架设基于 PHP 的 Web 服务器。采用 Linux+Apache+PHP+MySQL 这种开源免费的框架结构可以为网站经营者节省很大一笔开支。

7. 版本更新速度快

与数年才更新一次的 ASP 相比，PHP 的更新速度要快得多，因为 PHP 几乎每年更新一次。

8. 模板化

实现程序逻辑与用户界面分离。

9. 应用范围广

目前在互联网有很多网站的开发都是通过 PHP 语言来完成的，如搜狐、网易和百度等，在这些知名网站的创作开发中都应用到了 PHP 语言。

21.1.3 PHP 的工作原理

PHP 是基于服务器端运行的脚本程序语言，实现数据库和网页之间的数据交互。

一个完整的 PHP 系统由以下几个部分构成。

（1）操作系统：网站运行服务器所使用的操作系统。PHP 不要求操作系统的特定性，其跨平台的特性允许 PHP 运行在任何操作系统上，如 Windows 和 Linux 等。

（2）服务器：搭建 PHP 运行环境时所选择的服务器。PHP 支持多种服务器软件，包括 Apache、IIS 等。

（3）PHP 包：实现对 PHP 文件的解析和编译。

（4）数据库系统：实现系统中数据的存储。PHP 支持多种数据库系统，包括 MySQL、SQL Server、



Oracle 及 DB2 等。

（5）浏览器：浏览网页。由于 PHP 在发送到浏览器的时候已经被解析器编译成其他的代码，所以 PHP 对浏览器没有任何限制。

在图 21.1 中，完整地展示了用户通过浏览器访问 PHP 网站系统的全过程，可以更加清晰地理清它们之间的关系。

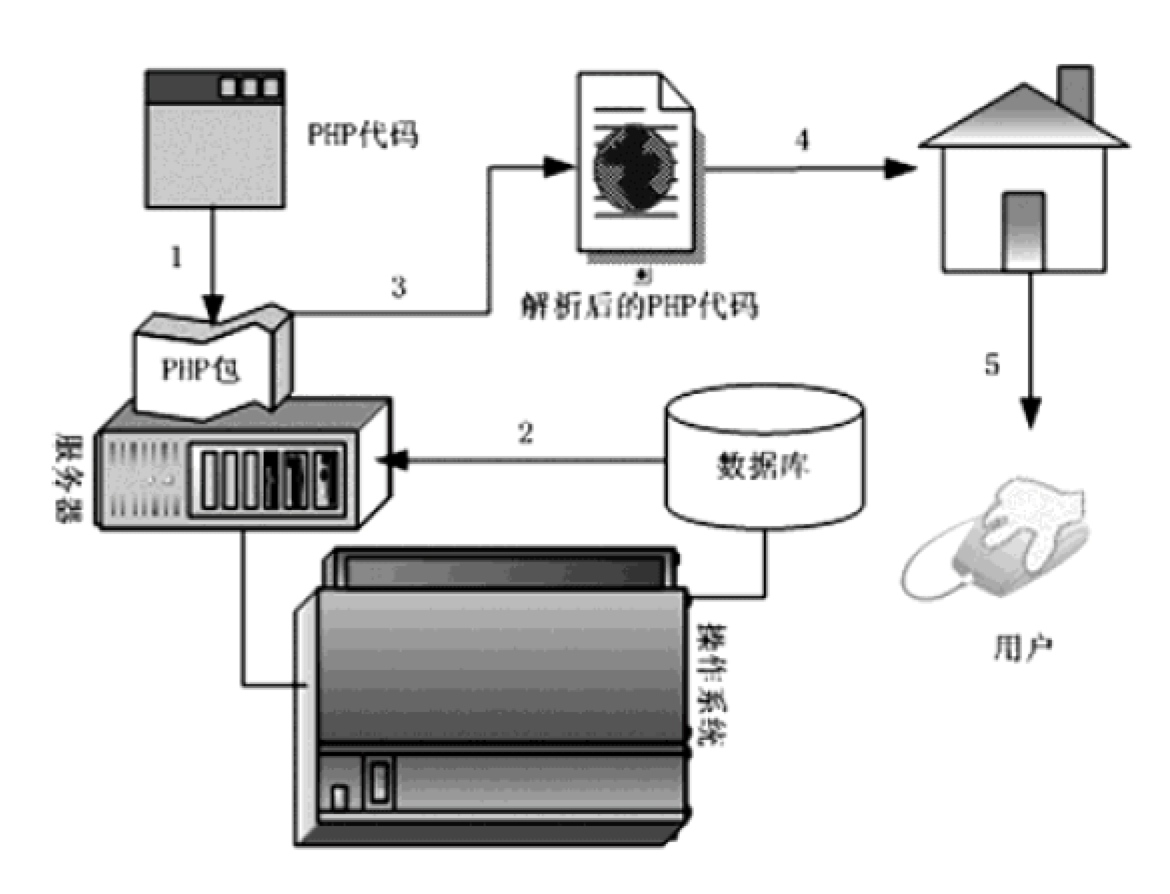


图 21.1　PHP 的工作原理



PHP 工作原理如下。

（1）PHP 的代码传递给 PHP 包，请求 PHP 包进行解析并编译。

（2）服务器根据 PHP 代码的请求读取数据库。

（3）服务器与 PHP 包共同根据数据库中的数据或其他运行变量，将 PHP 代码解析成普通的 HTML 代码。

（4）解析后的代码发送给浏览器，浏览器对代码进行分析获取可视化内容。

（5）用户通过访问浏览器浏览网站内容。在 Android 4.0 中，采用默认的主题（Theme.Holo）时，android:prompt 属性看不到具体的效果，但是采用 Theme.Black 时，就可以看到在弹出的下拉框上将显示该标题。

21.1.4　PHP 结合数据库应用的优势

在实际应用中，PHP 的一个最常见的应用就是与数据库结合。无论是建设网站还是设计信息系统，都少不了数据库的参与。广义的数据库可以理解成关系型数据库管理系统、XML 文件，甚至文本文件等。

PHP 支持多种数据库，而且提供了与诸多数据库连接的相关函数或类库。一般来说，PHP 与 MySQL 是比较流行的一个组合。该组合的流行不仅是因为它们都可以免费获取，更多的是因为 PHP 内部对

MySQL 数据库的完美支持。

当然，除了使用 PHP 内置的连接函数以外，还可以自行编写函数来间接存取数据库。这种机制给程序员带来了很大的灵活性。

21.2　PHP 操作 MySQL 数据库的基本步骤

与其他语言类似，PHP 操作 MySQL 数据库的过程一般分为 5 步，分别为连接 MySQL 数据库服务器、选择数据库、执行 SQL 语句、关闭结果集以及断开与 MySQL 服务器的连接，如图 21.2 所示。

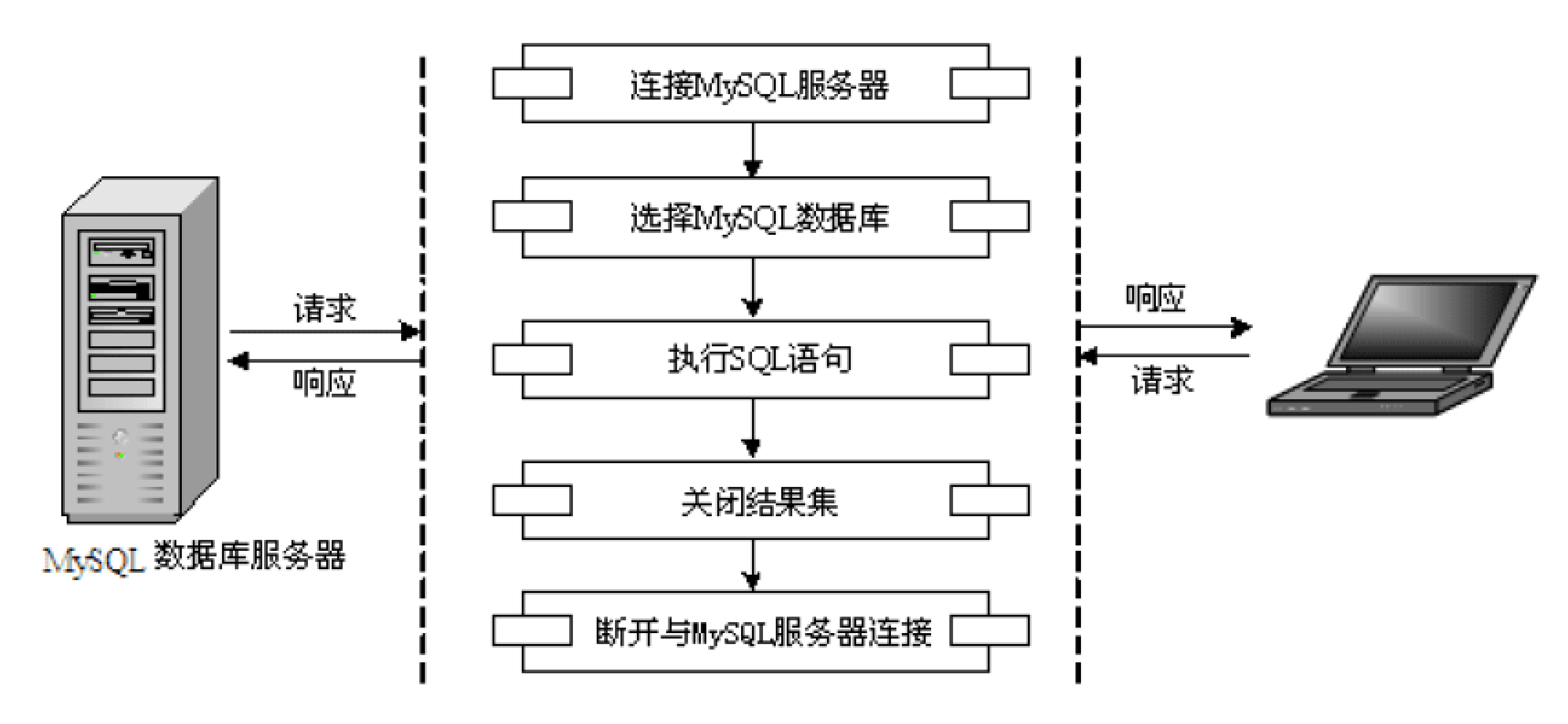


图 21.2　PHP 操作 MySQL 数据库的步骤

下面将对图 21.2 中的 5 个步骤进行具体介绍。

（1）连接 MySQL 服务器。

应用 mysql_connect()函数建立与 MySQL 服务器的连接，并返回一个连接标识，在以后对 MySQL 服务器进行操作时，可以根据这个连接标识定位不同的连接。

（2）选择数据库。

应用 mysql_select_db()函数选择 MySQL 数据库服务器上的数据库，并与该数据库建立连接。

（3）执行 SQL 语句。

在选择的数据库中应用 mysql_query()函数执行 SQL 语句。对数据的操作主要包括以下 5 种方式。

① 查询数据：应用 SELECT 语句实现数据的查询功能。

② 显示数据：应用 SELECT 语句显示数据的查询结果。

③ 插入数据：应用 INSERT 语句向数据库中插入数据。

④ 更新数据：应用 UPDATE 语句修改数据库中的记录。

⑤ 删除数据：应用 DELETE 语句删除数据库中的记录。

（4）关闭结果集。

数据库操作完成后，需要关闭结果集，以释放系统资源。

```
mysql_free_result($result);
```



技巧

如果在多个网页中都要频繁进行数据库访问，则可以建立与数据库服务器的持续连接来提高效率。因为每次与数据库服务器的连接需要较长的时间和较大的资源开销，持续的连接相对来说会更有效。建立持续连接的方法就是在数据库连接时，调用函数 mysql_pconnect()代替 mysql_connect()函数。建立的持续连接在本程序结束时，不需要调用 mysql_close()来关闭。下次程序在此执行 mysql_pconnect()函数时，系统自动直接返回已经建立的持续连接的 ID 号，而不再去真的连接数据库。

（5）断开与 MySQL 服务器的连接。

每使用一次 mysql_connect()或 mysql_query()函数，都会消耗系统资源。这在少量用户访问 Web 网站时影响不明显，但如果用户连接超过一定数量，就会造成系统性能的下降，甚至死机。为了避免这种现象的发生，在完成数据库的操作后，可以应用 mysql_close()函数关闭与 MySQL 服务器的连接，以节省系统资源。

21.3　使用 PHP 操作 MySQL 数据库

根据 21.2 节中介绍的 PHP 操作 MySQL 数据库的步骤，下面详细讲解每个步骤是如何实现的，都应用了哪些函数和方法。

21.3.1　应用 mysql_connect()函数连接 MySQL 服务器

PHP 操作 MySQL 数据库，首先要建立与 MySQL 数据库的连接，PHP 实现与数据库连接相对简便，只需使用 mysql_connect()函数即可，函数语法如下。

```
resource mysql_connect ( [string server [, string username [, string password [, bool new_link [, int client_flags]]]]] )
```

mysql_connect()函数用于打开一个到 MySQL 服务器的连接，如果成功则返回一个 MySQL 连接标识，失败则返回 false。该函数的参数如表 21.1 所示。

表 21.1　mysql_connect()函数的参数说明

参　数	说　明
server	MySQL 服务器。可以包括端口号，如“hostname:port”；或者到本地套接字的路径，如对于 localhost 的“:/path/to/socket”。如果 PHP 指令 mysql.default_host 未定义（默认情况），则默认值是“localhost:3306”
username	用户名。默认值是服务器进程所有者的用户名
password	密码。默认值是空密码

续表

参　　数	说　　明
new_link	如果用同样的参数再次调用 mysql_connect()函数，将不会建立新连接，而将返回已经打开的连接标识。参数 new_link 改变此行为并使 mysql_connect()函数总是打开新的连接，即使 mysql_connect()函数曾在前面被用同样的参数调用过
client_flags	client_flags 参数可以是以下常量的组合：MYSQL_CLIENT_SSL，MYSQL_CLIENT_COMPRESS，MYSQL_CLIENT_IGNORE_SPACE 或 MYSQL_CLIENT_INTERACTIVE

例如，使用 mysql_connect()函数连接本地 MySQL 服务器，代码如下。

```
<?php
$conn = mysql_connect("localhost", "root", "root") or die("连接数据库服务器失败！".mysql_error());
?>
```

例 21.1　应用 mysql_connect()函数创建与 MySQL 服务器的连接，MySQL 数据库服务器地址为 127.0.0.1，用户名为 root，密码为 root，代码如下。（**实例位置：光盘\TM\sl\21\21.1**）

```
<?php
$host = "127.0.0.1";                //MySQL 服务器地址
$userName = "root";                 //用户名
$password = "root";                 //密码
if ($connID = mysql_connect($host, $userName, $password)){
                                    //建立与 MySQL 数据库的连接，并弹出提示对话框
      echo "<script language='javascript'>alert('数据库连接成功！');</script>";
}else{
      echo "<script language='javascript'>alert('数据库连接失败！');</script>";
}
?>
```

运行上述代码，如果在本地计算机中安装了 MySQL 数据库，并且 root 用户名为 root，密码为 root，则会弹出如图 21.3 所示的对话框。



图 21.3　数据库连接成功

为了方便查询因为连接问题而出现的错误，采用 die()函数生成错误处理机制，使用 mysql_error()函数提取 MySQL 函数的错误文本。如果没有出错，则返回空字符串；如果浏览器显示“Warning: mysql_connect()…”的字样时，说明是数据库连接的错误，这样就能迅速地发现错误位置，及时改正。

说明

在 mysql_connect()函数前面添加符号“@”，用于限制这个命令的出错信息的显示。如果函数调用出错，将执行 or 后面的语句。die()函数表示向用户输出引号中的内容后，程序终止执行。这样是为了防止数据库连接出错时，用户看到一堆莫名其妙的专业名词，而是提示定制的出错信息。但在调试时不要屏蔽出错信息，避免出错后难以找到问题。

21.3.2　应用 mysql_select_db()函数选择 MySQL 数据库

成功与 MySQL 数据库建立连接后，需要选择 MySQL 数据库服务器中指定的数据库。PHP 中使用 mysql_select_db()函数实现数据库的选择功能，该函数的语法格式如下。

```
bool mysql_select_db ( string database_name [, resource link_identifier] )
```

mysql_select_db()函数用于设定与指定的连接标识符所关联的服务器上的当前激活数据库。如果没有指定连接标识符，则使用上一个打开的连接。如果没有打开的连接，本函数将无参数调用 mysql_connect()函数来尝试打开一个使用。其后的每个 mysql_query()函数调用都会作用于当前激活数据库。该函数的参数说明如表 21.2 所示。

表 21.2　mysql_select_db()函数的参数说明

参　　数	说　　明
database_name	必要参数，用户指定要选择的数据库名称
link_identifier	可选参数，数据库连接 ID，如果省略该参数，则默认为最近一次与数据库建立的连接

例如，与本地 MySQL 服务器中的 db_database21 数据库建立连接，代码如下。

```
<?php
$conn=mysql_connect("localhost","root","root");            //连接 mysql 数据库服务器
$select=mysql_select_db("db_database21",$conn);             //连接服务器中的 db_database21
if($select){                                                //判断是否连接成功
    echo "数据库连接成功！";
}
?>
```

例 21.2　首先使用 mysql_connect()函数建立与 MySQL 数据库的连接并返回数据库连接 ID，然后使用 mysql_select_db()函数选择 MySQL 数据库服务器中名为 db_database21 的数据库，实现代码如下。**（实例位置：光盘\TM\sl\21\21.2）**

```
<?php
$host = "127.0.0.1";                                        //MySQL 服务器地址
$userName = "root";                                         //用户名
$password = "root";                                         //密码
$dbName = "db_database21";                                  //数据库
$connID = mysql_connect($host, $userName, $password);       //建立与 MySQL 数据库服务器的连接
if(mysql_select_db($dbName, $connID)){                      //选择数据库
    echo "数据库选择成功！";
}else{
    echo "数据库选择失败！";
}
?>
```

运行上述代码，如果本地 MySQL 数据库服务器中存在名为 db_database21 的数据库，将在页面中显示如图 21.4 所示的提示信息。

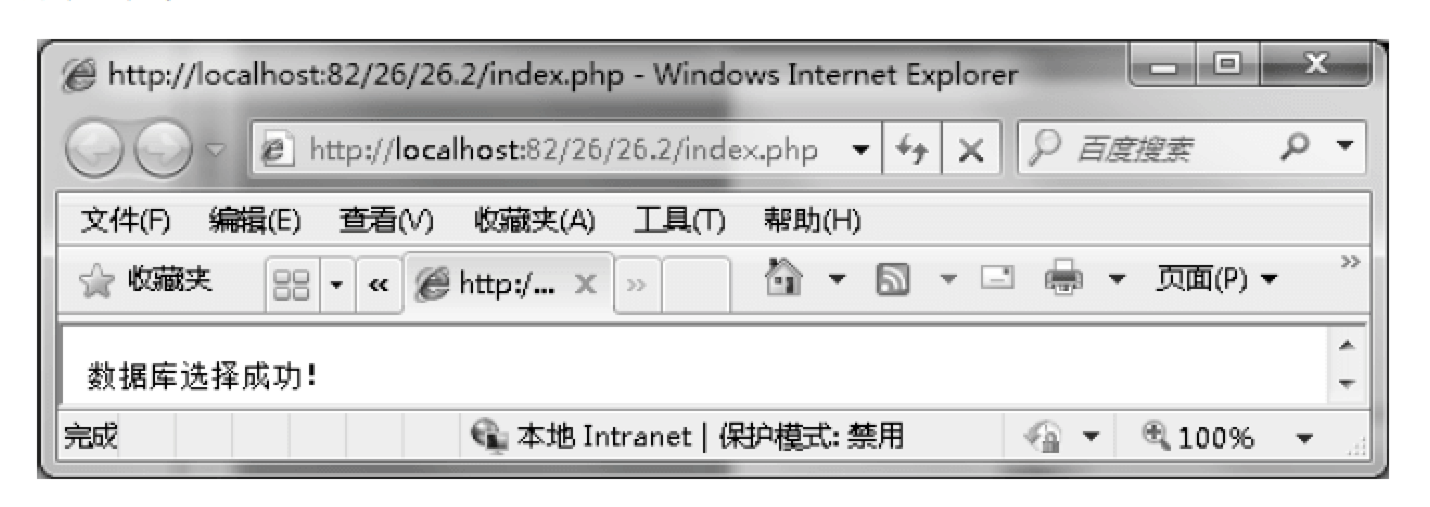


图 21.4　数据库选择成功

21.3.3　应用 mysql_query()函数执行 SQL 语句

成功选择 MySQL 数据库服务器中的数据库后，即可对所选数据库中的数据表进行查询、更改以及删除等操作，PHP 使用 mysql_query()函数就可以实现上述所有操作，操作极其简便，说明在 PHP 底层进行了复杂的封装，而提供给上层开发人员一种简便的编程模式，这也是 PHP 操作简便的体现和应用广泛的原因。mysql_query()函数的语法格式如下。

```
resource mysql_query ( string query [, resource link_identifier] )
```

mysql_query()函数用于执行一条查询语句，该函数的参数说明如表 21.3 所示。

表 21.3　mysql_query()函数的参数说明

参　数	说　明
query	字符串类型，传入的是 SQL 的指令，包括插入数据（insert）、修改记录（update）、删除记录（delete）、查询记录（select）
link_identfier	资源类型，传入的是由 mysql_connect()函数或 mysq1_pconnect()函数返回的连接号。如果省略该参数，则会使用最后一个打开的 MySQL 数据库连接

例如，向会员信息表 tb_user 中插入一条会员记录，SQL 语句的代码如下。

```
$result=mysql_query("insert into tb_user values('mr','root')",$conn);
```

例如，修改会员信息 tb_user 表中的会员记录，SQL 语句的代码如下。

```
$result=mysql_query("update tb_user set name='lx' where id='01'",$conn);
```

例如，删除会员信息 tb_user 表中的一条会员记录，SQL 语句的代码如下。

```
$result=mysql_query("delete from tb_user where name='mr'",$conn);
```

例如，查询会员信息 tb_user 表中 name 字段值为 mr 的记录，SQL 语句的代码如下。

```
$result=mysql_query("select * from tb_user where name='mr'",$conn);
```

上面的 SQL 语句代码都是将结果赋给变量$result。



例 21.3　查询学生信息表中学生的成绩信息，代码如下。（**实例位置：光盘\TM\sl\21\21.3**）

```
<?php
$host = "127.0.0.1";                                    //MySQL 数据库服务器
$userName = "root";                                     //用户名
$password = "root";                                     //密码
$dbName = "db_database21";                              //数据库名
$connID = mysql_connect($host, $userName, $password);   //连接 MySQL 数据库
mysql_select_db($dbName, $connID);                      //选择 MySQL 数据库
mysql_query("set names utf8");                          //设置字符集
echo "<table border=\"1px\" align=\"center\">
          <tr>
               <td>学号</td>
               <td>姓名</td>
               <td>班级</td>
               <td>语文</td>
               <td>数学</td>
               <td>英语</td>
          </tr>";
$query = mysql_query("select sno, sname, class, chinese, math ,english from tb_student", $connID);
                                                        //执行查询
while($result = mysql_fetch_array($query))              //获取结果集并输出查询结果
{
    echo   "<tr>
               <td>".$result["sno"]."</td>
               <td>".$result["sname"]."</td>
               <td>".$result["class"]."</td>
               <td>".$result["chinese"]."</td>
               <td>".$result["math"]."</td>
               <td>".$result["english"]."</td>
          </tr>";
}
echo "</table>";
?>
```

运行上述实例，结果如图 21.5 所示。

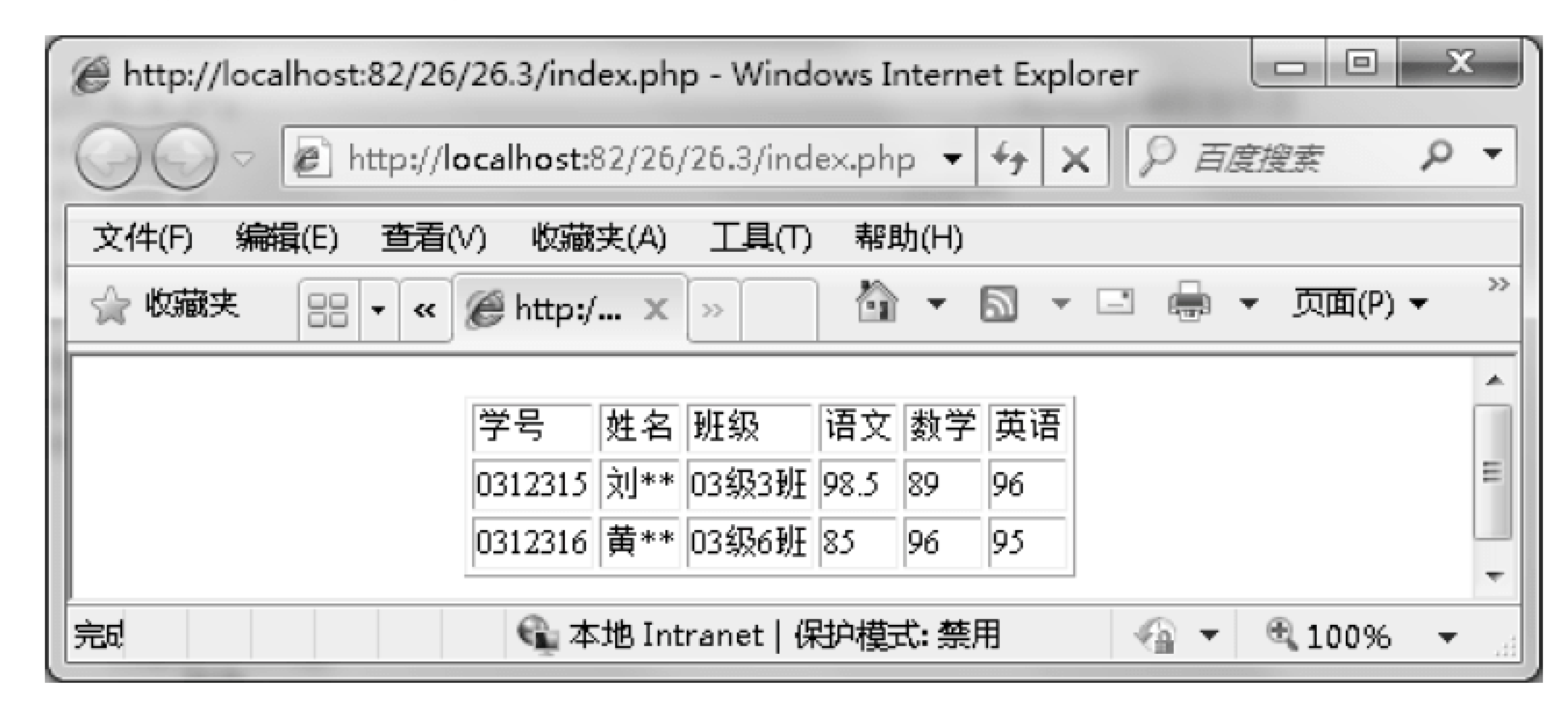


图 21.5　查询学生成绩

21.3.4　应用 mysql_fetch_array()函数将结果集返回到数组中

使用 mysql_query()函数执行 SELECT 语句时，成功将返回查询结果集，返回结果集后，使用 mysql_fetch_array()函数可以获取查询结果集信息，并放入到一个数组中，函数语法如下。

```
array mysql_fetch_array ( resource result [, int result_type] )
```

其中，参数 result 是资源类型的参数，要传入的是由 mysql_query()函数返回的数据指针；参数 result_type 是可选项，设置结果集数组的表述方式，默认值是 MYSQL_BOTH。其可选值如下。

（1）MYSQL_ASSOC：表示数组采用关联索引。

（2）MYSQL_NUM：表示数组采用数字索引。

（3）MYSQL_BOTH：同时包含关联和数字索引的数组。

例 21.4　按员工编号以模糊查询的方式查询员工信息，并显示全部查询结果。**（实例位置：光盘\TM\sl\21\21.4）**

具体实现步骤如下。

（1）建立与 MySQL 数据库的连接，并返回数据库连接 ID，代码如下。

```
<?php
$connID=mysql_connect("localhost","root","root");            //建立与数据库的连接
mysql_select_db("db_database21", $connID);                   //选择数据库
mysql_query("set names utf8");                               //设置字符集
?>
```

（2）建立查询信息录入表单，表单及表单元素如表 21.4 所示。

表 21.4　员工信息录入表单

元素类型	元素名称	属性设置	说明
表单	form1	name="form1" method="post" action="<?php echo $_SERVER['PHP_SELF']?>"	表单
文本域	number	name="number" type="text" id="number"	录入员工编号
隐藏域	flag	type="hidden" name="flag" value="1"	判断表单是否提交
提交按钮	submit	name="submit" type="submit" value="提交"	“提交”按钮

（3）使用$_POST 全局数组接收表单提交的 flag 元素的值，并使用 isset()函数判断是否已经设置了该元素的值，如果已设置则说明已经提交了表单，然后采用模糊查询的方式查询所有与查询关键字相匹配的员工信息，并使用 while 循环将查询结果显示出来，代码如下。

```
<?php
  if(isset($_POST["flag"])){
        $query=mysql_query("select * from tb_employee where number like '%".$_POST["number"]."%'");
        if($query){
              while($myrow=mysql_fetch_array($query)){
?>
```



```
  <tr>
    <td    align="center"    bgcolor="#FFFFFF"    class="STYLE4"><span    class="STYLE2"><?php    echo
$myrow[number];? > </span></td>
    <td    align="center"    bgcolor="#FFFFFF"    class="STYLE4"><span    class="STYLE2"><?php    echo
$myrow[name];?>    </span></td>
    <td height="23" align="center" bgcolor="#FFFFFF" class="STYLE4"><span class="STYLE2"><?php echo
$myrow [tel];?></span></td>
    <td height="23" align="center" bgcolor="#FFFFFF" class="STYLE4"><span class="STYLE2"><?php echo
$myrow [address];?></span></td>
  </tr>
 <?php
          }
      }
  }
?>
```

运行该实例，在员工查询信息录入表单中输入员工编号，然后单击“提交”按钮，即可以模糊查询的方式查询出所有与查询关键字相匹配的员工信息，如图 21.6 所示。

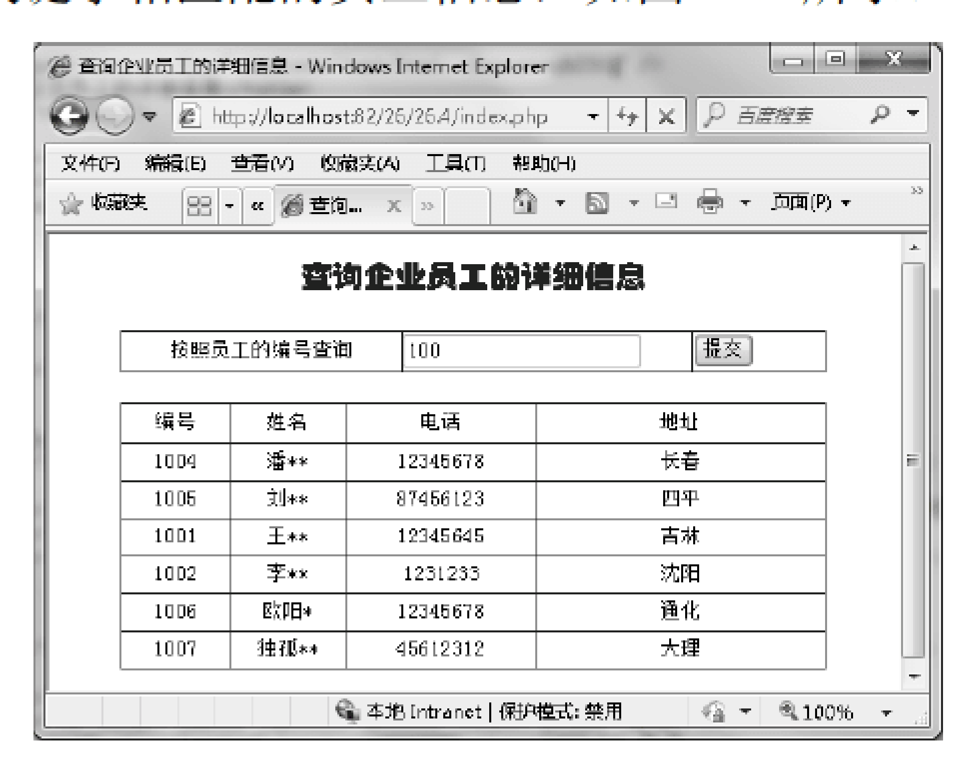


图 21.6　查询员工信息

21.3.5　应用 mysql_fetch_object()函数从结果集中获取一行作为对象

21.3.4 节中讲解了应用 mysql_fetch_array()函数来获取结果集中的数据。除了这个方法以外，应用 mysql_fetch_object()函数也可以轻松实现这一功能，下面通过同一个实例的不同方法来体验一下这两个函数在使用上的区别。首先介绍 mysql_fetch_object()函数。语法如下。

```
object   mysql_fetch_object ( resource result )
```

mysql_fetch_object()函数和 mysql_fetch_array()函数类似，只有一点区别：即前者返回一个对象而不是数组，即该函数只能通过字段名来访问数组。访问结果集中行的元素的语法结构如下。

```
$row->col_name                    //col_name 为列名，$row 代表结果集
```

例如，如果从某数据表中检索 id 和 name 值，可以用$row->id 和$row-> name 访问行中的元素值。



本函数返回的字段名是区分大小写的，这是初学者学习时最容易忽视的问题。

例 21.5　使用 mysql_fetch_object()函数获取查询到图书的信息。（**实例位置：光盘\TM\sl\21\21.5**）

具体开发步骤如下。

（1）建立图书查询表单，表单及表单元素说明如表 21.5 所示。

表 21.5　图书信息查询表单及表单说明

元素类型	元素名称	属性设置	说　　明
表单	myform	name="myform" method="post" action=""	表单
文本域	txt_book	name="txt_book" type="text" id="txt_book" size="25"	查询关键字
提交按钮	submit	type="submit" name="Submit" value="查询"	"查询"按钮

（2）建立与 MySQL 数据库的连接、设置字符集，并返回数据库连接 ID，代码如下。

```
$link=mysql_connect("localhost","root","root") or die("数据库连接失败".mysql_error()); //建立与数据库的连接
mysql_select_db("db_database21",$link);                                    //选择数据库
mysql_query("set names gb2312");                                           //设置字符集
```

（3）应用 mysql_query()函数执行 SQL 查询语句，并使用 mysql_fetch_object()函数获取查询语句的结果集，代码如下。

```
if ($_POST[Submit]=="查询"){
    $txt_book=$_POST[txt_book];                    //接收查询关键字
    $sql=mysql_query("select * from tb_book where bookname like '%".trim($txt_book)."%'");
                                                   //如果选择的条件为"like"，则进行模糊查询
     $info=mysql_fetch_object($sql);
}
```

（4）应用 do…while 循环语句以对象的方式输出结果集中的图书信息到浏览器中，代码如下。

```
do{
?>
<tr align="left" bgcolor="#FFFFFF">
  <td height="20" align="center"><?php echo $info->id; ?></td>
  <td > <?php echo $info->bookname; ?></td>
  <td align="center"><?php echo $info->issuDate; ?></td>
  <td align="center"><?php echo $info->price; ?></td>
  <td align="center"> <?php echo $info->maker; ?></td>
  <td> <?php echo $info->publisher; ?></td>
</tr>
<?php
}while($info=mysql_fetch_object($sql));
```

保存 index.php 动态页，在 IE 浏览器地址栏中输入地址，按 Enter 键，运行结果如图 21.7 所示。

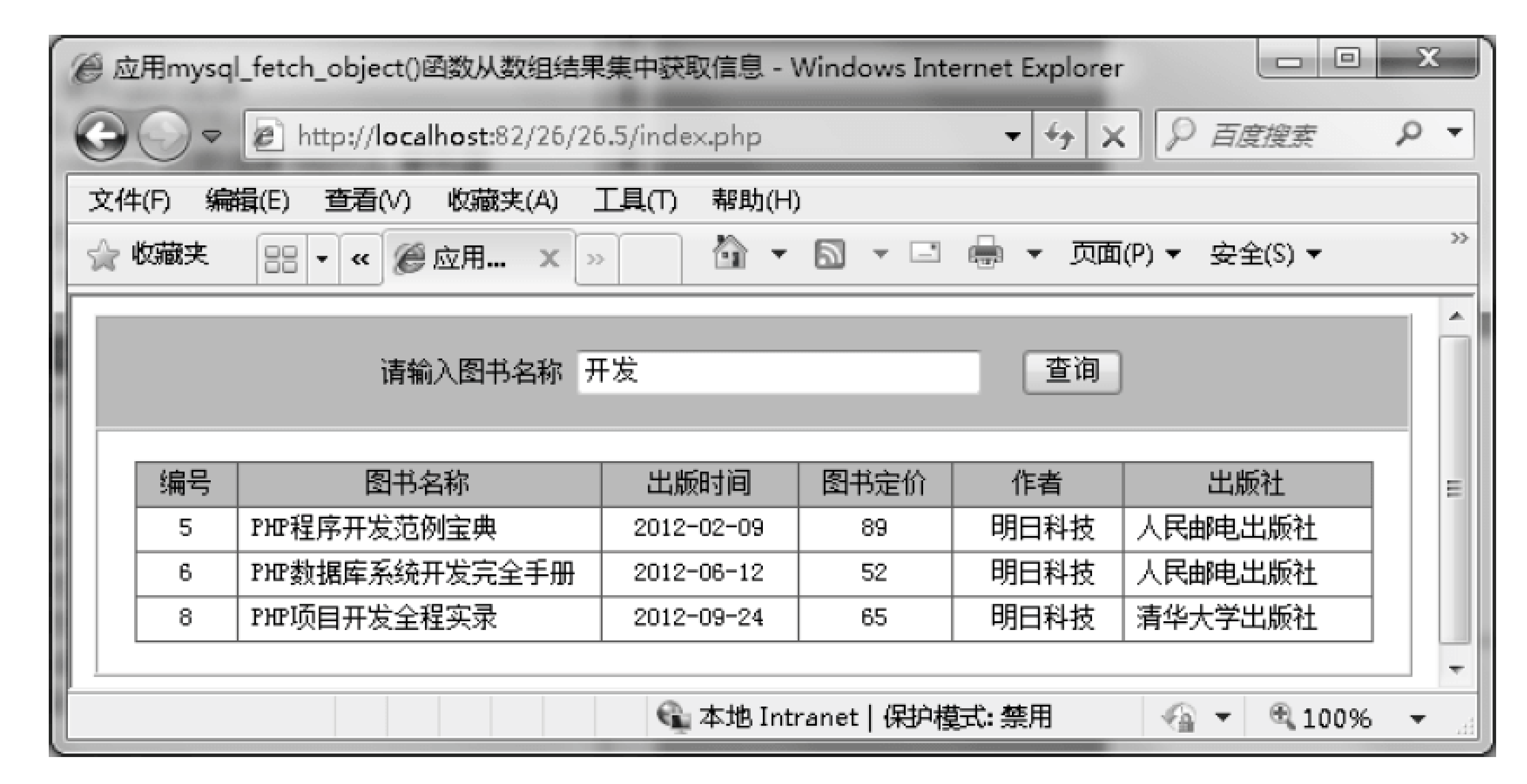


图 21.7　查询图书信息

21.3.6　应用 mysql_fetch_row()函数从结果集中获取一行作为枚举数组

mysql_fetch_row()函数从结果集中取得一行作为枚举数组。在应用 mysql_fetch_row()函数逐行获取结果集中的记录时，只能使用数字索引来读取数组中的数据，其语法如下。

```
array mysql_fetch_row ( resource result )
```

mysql_fetch_row()函数返回根据所取得的行生成的数组，如果没有更多行则返回 FALSE。返回数组的偏移量从 0 开始，即以$row[0]的形式访问第一个元素（只有一个元素时也是如此）。

例 21.6　查询图书信息，并使用 mysql_fetch_row()函数获取结果集显示图书信息。**（实例位置：光盘\TM\sl\21\21.6）**

具体开发步骤如下。

（1）创建项目、添加表单、连接 MySQL 服务器以及设置默认数据库的实现过程与例 21.5 开发步骤中的（1）、（2）相同，这里不再赘述。

（2）在应用 mysql_query()函数执行 SQL 查询语句后，与例 21.5 不同的是，本实例使用 mysql_fetch_row()函数获取查询语句的结果集。代码如下。

```
<?php
$sql=mysql_query("select * from tb_book");
$row=mysql_fetch_row ($sql);
if ($_POST[Submit]=="查询"){
$txt_book=$_POST[txt_book];
//如果选择的条件为“like”，则进行模糊查询
$sql=mysql_query("select * from tb_book where bookname like '%". trim($txt_book)."%'");
$row=mysql_fetch_row($sql);
}
?>
```

（3）应用 if 条件语句对结果集变量$info 进行判断，如果该值为假，则检索的图书信息不存在，

应用 echo 语句输出提示信息，代码如下。

```
<?php
if($row==false){              //如果检索的信息不存在，则输出相应的提示信息
     echo "<div align='center' style='color:#FF0000; font-size:12px'>对不起，您检索的图书信息不存在!</div>";
}
?>
```

（4）应用 do…while 循环语句以对象的方式输出结果集中的图书信息到浏览器中，代码如下。

```
<?php
do{
?>
<tr align="left" bgcolor="#FFFFFF">
  <td height="20" align="center"><?php echo $row[0]; ?></td>
  <td > <?php echo $row[1]; ?></td>
  <td align="center"><?php echo $row[2]; ?></td>
  <td align="center"><?php echo $row[3]; ?></td>
  <td align="center"> <?php echo $row[4]; ?></td>
  <td> <?php echo $row[5]; ?></td>
</tr>
<?php
}while($row=mysql_fetch_row($sql));
?>
```

保存 index.php 动态页，在 IE 浏览器地址栏中输入地址，按 Enter 键，运行结果如图 21.8 所示。

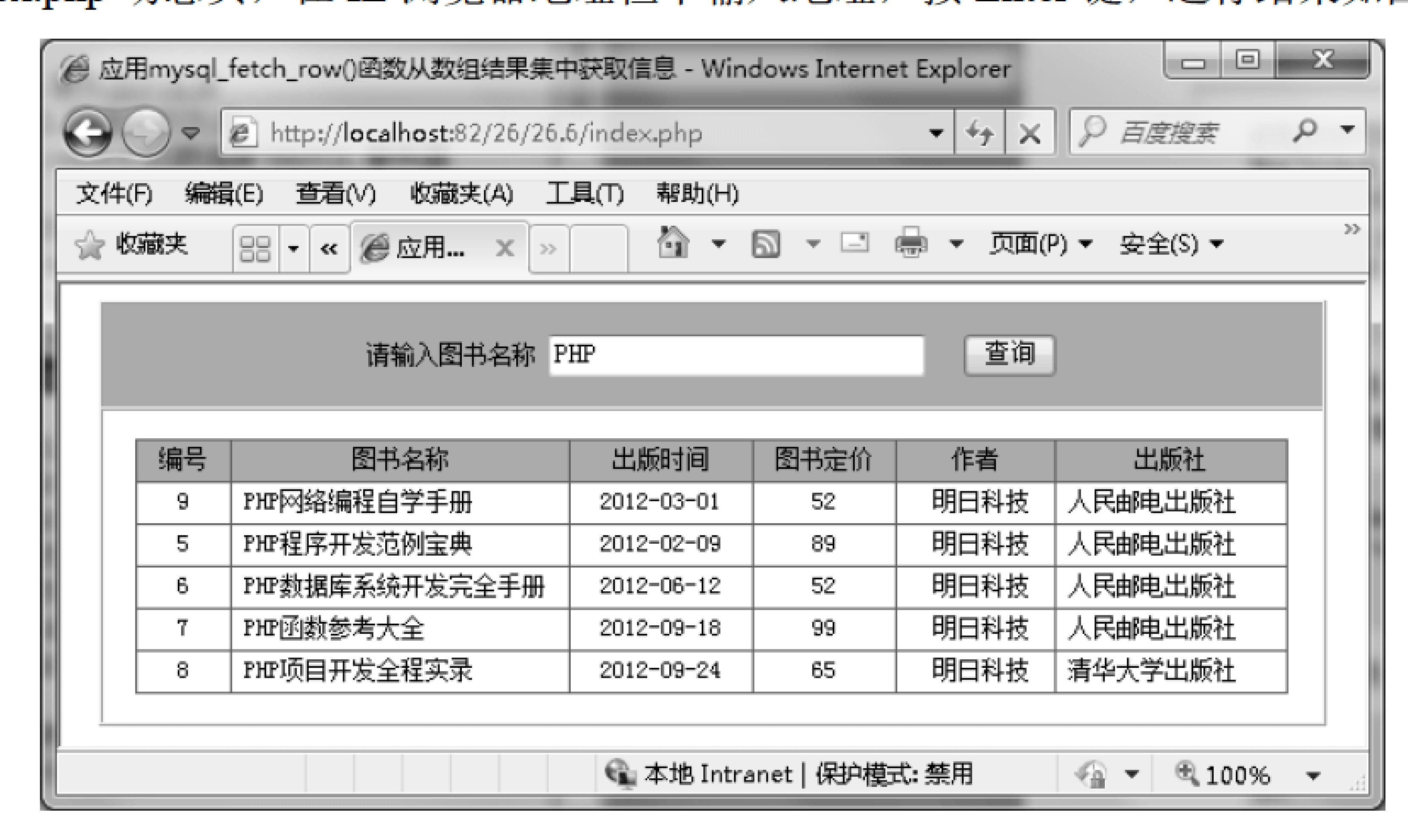


图 21.8　使用 mysql_fetch_row()函数获取结果集查询图书信息

21.3.7　应用 mysql_num_rows()函数获取查询结果集中的记录数

使用 mysql_num_rows()函数可以获取由 select 语句查询到的结果集中行的数目，mysql_num_rows()



函数的语法如下。

```
int mysql_num_rows ( resource result )
```

此命令仅对 SELECT 语句有效。要取得被 INSERT、UPDATE 或者 DELETE 语句所影响到的行的数目，要使用 mysql_affected_rows()函数。

例 21.7　使用 mysql_num_rows()函数获取查询结果的记录数。**（实例位置：光盘\TM\sl\21\21.7）**

具体开发步骤如下。

（1）建立与 MySQL 数据库的连接，设置字符集为 GB2312，并返回数据库连接 ID，代码如下。

```
$link=mysql_connect("localhost","root","root") or die("数据库连接失败".mysql_error());      //建立连接
mysql_select_db("db_database21",$link);                                                    //选择数据库
mysql_query("set names gb2312");                                                           //设置字符集
```

（2）默认情况下显示所有图书信息，代码如下。

```
$sql=mysql_query("select * from tb_book");          //查询所有图书信息
$info=mysql_fetch_object($sql);                     //获取结果集
```

（3）如果用户输入了查询关键字，并单击了“查询”按钮，则采用模糊查询的方式显示出所有符合条件的记录，代码如下。

```
if ($_POST[Submit]=="查询"){
    $txt_book=$_POST[txt_book];
    $sql=mysql_query("select * from tb_book where bookname like '%".trim($txt_book)."%'");
    //如果选择的条件为“like”，则进行模糊查询
    $info=mysql_fetch_object($sql);
}
if($info==false){              //如果检索的信息不存在，则输出相应的提示信息
     echo "<div align='center' style='color:#FF0000; font-size:12px'>对不起，您检索的图书信息不存在!</div>";
}
do{
?>
<tr align="left" bgcolor="#FFFFFF">
    <td height="20" align="center"><?php echo $info->id; ?></td>
    <td > <?php echo $info->bookname; ?></td>
    <td align="center"><?php echo $info->issuDate; ?></td>
    <td align="center"><?php echo $info->price; ?></td>
    <td align="center"> <?php echo $info->maker; ?></td>
    <td> <?php echo $info->publisher; ?></td>
</tr>
<?php
    }while($info=mysql_fetch_object($sql));
?>
```

在 IE 浏览器地址栏中输入地址，按 Enter 键，程序默认输出图书信息表中的全部图书信息，并自动汇总记录条数，如图 21.9 所示。在文本框中输入要检索的图书名称，如“开发”（支持模糊查询，程序自动去除查询关键字左右空格），单击“查询”按钮，即可按条件检索指定的图书信息到浏览器，并自动汇总检索到的记录条数，运行结果如图 21.10 所示。

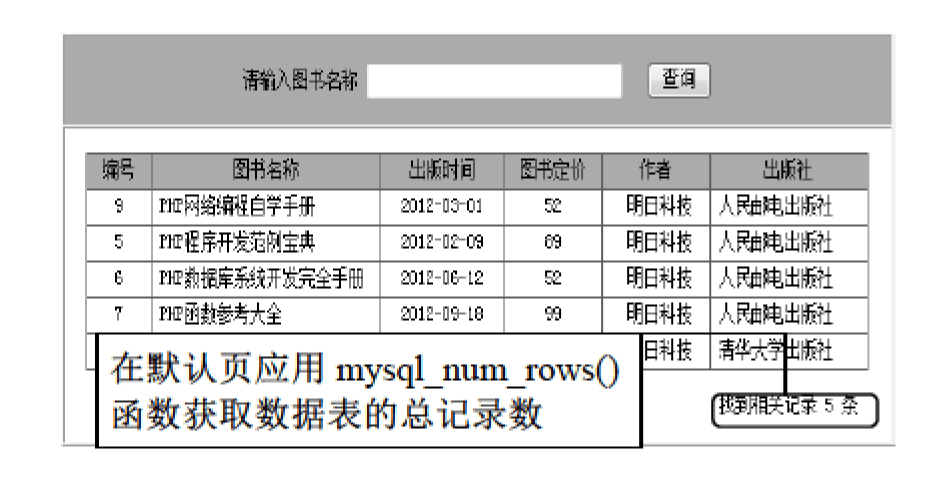


图 21.9　默认统计数据表中所有记录

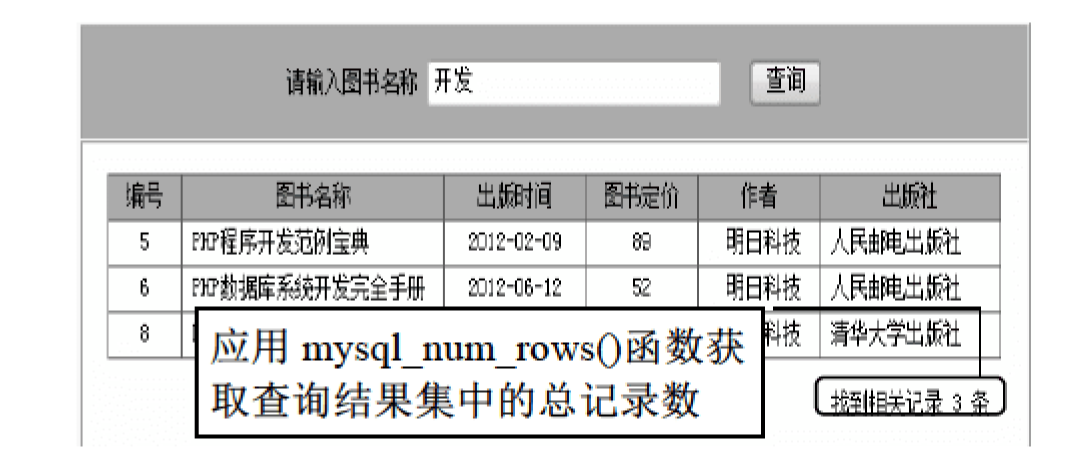


图 21.10　应用 mysql_num_rows()函数获取查询结果集中的记录数



技巧

如果要获取由 insert、update、delete 语句所影响到的数据行数，则必须应用 mysql_affected_rows()函数来实现。

21.3.8　应用 mysql_free_result()函数释放内存

mysql_free_result()函数用于释放内存，数据库操作完成后，需要关闭结果集，以释放系统资源，该函数的语法如下。

```
mysql_free_result($result);
```

mysql_free_result()函数将释放所有与结果标识符 result 所关联的内存。该函数仅需要在考虑到返回很大的结果集时会占用多少内存时调用。在脚本结束后所有关联的内存都会被自动释放。

21.3.9　应用 mysql_close()函数关闭连接

每使用一次 mysql_connect()或 mysql_query()函数，都会消耗系统资源。在少量用户访问 Web 网站时问题还不大，但如果用户连接超过一定数量时，就会造成系统性能的下降，甚至死机。为了避免这种现象的发生，在完成数据库的操作后，应使用 mysql_close()函数关闭与 MySQL 服务器的连接，以节省系统资源。mysql_close()函数的语法如下。

```
mysql_close($conn);
```

在 Web 网站的实际项目开发过程中，经常需要在 Web 页面中查询数据信息。查询后使用 mysql_close()函数关闭数据源。

例 21.8　使用 mysql_connect()函数建立与数据库的连接，然后使用 mysql_select_db()函数选择数据库并使用 mysql_close()函数断开与 MySQL 数据库的连接，在断开与 MySQL 数据库连接后，再次使用 mysql_select_db()函数选择数据库，从两次选择数据库的情况来判断 mysql_close()函数是否能起到断开数据库连接的作用。实例代码如下。**（实例位置：光盘\TM\sl\21\21.8）**

```
<?php
$host = "127.0.0.1";                                          //MySQL 数据库服务器
```



```
$userName = "root";                                      //用户名
$password = "root";                                      //密码
$dbName = "db_database21";                               //数据库名
$connID = mysql_connect($host, $userName, $password);   //连接 MySQL 数据库
mysql_select_db($dbName, $connID);                       //选择数据库
mysql_close($connID);                                    //断开与数据库连接
mysql_select_db($dbName, $connID);                       //再次选择数据库
?>
```

运行本实例，将在页面中输出如图 21.11 所示的错误信息，从错误信息中可以判断，第一次对数据库选择操作是成功的，而第二次操作是失败的，即可证明 mysql_close()函数成功关闭了与数据库的连接。

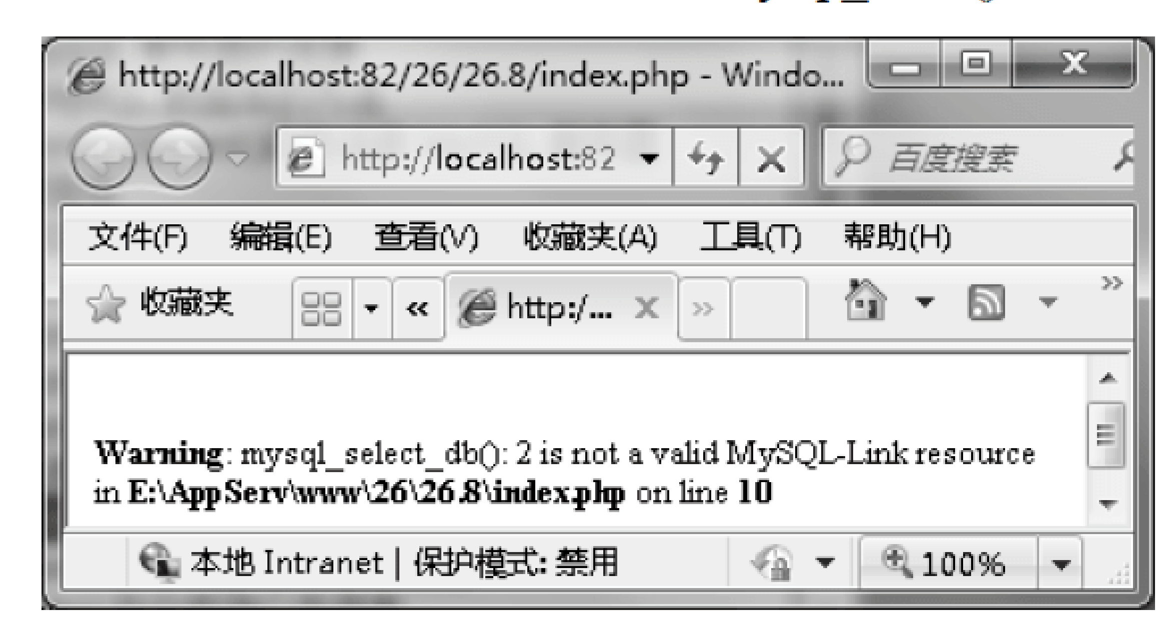


图 21.11　错误提示

21.4　PHP 管理 MySQL 数据库中的数据

管理 MySQL 数据库中的数据主要是对数据进行添加、修改、删除、查询等操作，只有熟练地掌握这部分知识才能够独立开发出基于 PHP 的数据库应用。

21.4.1　向数据库中添加数据

向数据库中添加数据主要通过 mysql_query()函数和 INSERT 语句实现。

例 21.9　发表新闻，填写新闻标题及新闻内容，当用户单击“提交”按钮时，判断新闻标题及内容是否为空，如果不为空则将数据添加到数据库中，关键代码如下。**（实例位置：光盘\TM\sl\21\21.9）**

```
<?php
$conn=mysql_connect("localhost","root","root");
mysql_select_db("db_database21",$conn);
mysql_query("set names uft8");
if(isset($_POST['submit'])   and $_POST['name']!=null and $_POST['news']!=null and $_POST['submit']=="提交
"){
      $insert=mysql_query("insert into tb_news(name,news) values('".$_POST['name']."','".$_POST['news']."')");
      if($insert){
      echo "<script> alert('发表成功!'); window.location.href='index.php'</script>";
```

```
        }else{
        echo "<script> alert('发表失败!'); window.location.href='index.php'</script>";
        }
}else{
        echo "<script> alert('发表失败!'); window.location.href='index.php'</script>";
}
?>
```

运行结果如图 21.12 所示。

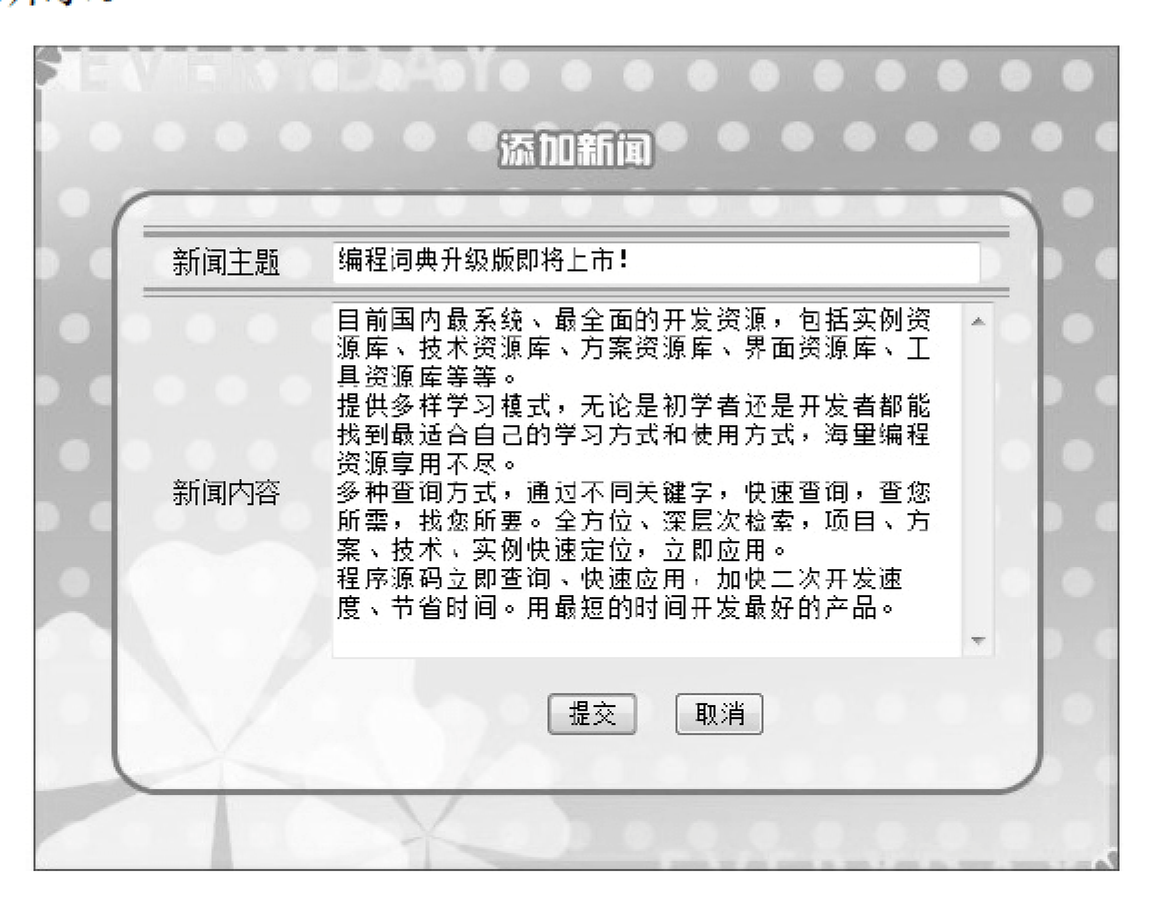


图 21.12　添加新闻

例 21.10　将文件以二进制的形式上传到数据库中，并且输出上传的数据。运行结果如图 21.13 所示。**（实例位置：光盘\TM\sl\21\21.10）**



图 21.13　将数据以二进制形式上传到数据库



（1）通过 Dreamweaver 开发工具创建一个 index.php 页，添加一个表单，设置表单的 method 属性值为 post，设置表单的 action 属性值为 index_ok.php；添加表单元素，提交图书信息；添加“提交”按钮，其关键代码如下。

```
<form name="form" method="post" action="index_ok.php" onSubmit="return check_form(this)">
<input name="cover" type="file" id="cover" size="30">
<input type="submit" name="Submit" value="提交">
</form>
```

（2）创建 conn 文件夹，编写 conn.php 文件，连接 MySQL 数据库服务器，连接 db_database21 数据库，设置数据库变量编码格式为 utf8。

（3）创建 index_ok.php 文件，通过$_POST[]方法获取表单中提交的数据，并判断上传图片的格式是否正确，通过 fopen()函数和 fread()函数读取表单中提交的图片数据，编写 insert 语句将数据添加到指定的数据表中，其关键代码如下。

```
<?php
include ("conn/conn.php");                                          //连接数据库
$bookname = $_POST["bookname"];                                     //获取表单中提交的数据
$price = $_POST["price"];
$maker = $_POST["maker"];
$issuDate = date("Y-m-d H:i:s");
$publisher = $_POST["publisher"];
$synopsis = $_POST["synopsis"];
if ($_POST["Submit"] == true) {
    $cover = $_FILES["cover"]['name'];                              //获取表单中提交的图片
    $cover_type = strstr($cover, ".");                              //获取从"."到最后的字符
    if ($cover_type != ".jpg" && $cover_type != ".gif" && $cover_type !=
     ".JPG" && $cover_type != ".GIF" && $cover_type != ".bmp" && $cover_type !=
     ".BMP") {                                                      //判断图片的格式
        echo  "<script>alert('封面图片格式不对，请进行处理后在上传！');  window.location.href='index.php';
</script>";
    } else {
        $cover=iconv("utf-8","gb2312",$cover);                      //设置字符串的编码格式
        $path = "uploadfiles/".$cover;
        @move_uploaded_file($_FILES["cover"]["tmp_name"],$path);
        $fp = fopen($path, "rb");                                   //以二进制形式打开图片
        $image = addslashes(
        @fread($fp, filesize($path)));                              //读取二进制的数据
//将数据添加到指定的数据表中
        $sql  =  "insert  into  tb_book(bookname,price,maker,issuDate,publisher,synopsis,cover)values('$bookname',
'$price','$maker','$issuDate','$publisher','$synopsis','$image')";
        $result = mysql_query($sql,
        $conn);
        if ($result == true) {
            echo iconv("utf-8","gb2312","文件上传成功!!");
            echo "<meta http-equiv=\"refresh\" content=\"30 url=index.php\">";
        } else {
            echo iconv("utf-8","gb2312","文件上传失败!!");
```

```
                echo "<meta http-equiv=\"refresh\" content=\"30 url=index.php\">";
            }
        }
}
?>
```

21.4.2 浏览数据库中的数据

浏览数据库中的数据通过 mysql_query()函数和 select 语句查询数据，使用 mysql_fetch_assoc()函数将查询结果返回到数组中。

例 21.11 浏览 tb_news 表中的新闻信息，具体代码如下。（**实例位置：光盘\TM\sl\21\21.11**）

```
<?php
/*连接数据库*/
$conn=mysql_connect("localhost","root","root");            //连接数据库服务器
mysql_select_db("db_database21",$conn);                    //选择数据库
mysql_query("set names uft8");                             //设置编码格式
$arr=mysql_query("select * from tb_news",$conn);           //执行查询语句
/*使用 while 语句循环 mysql_fetch_assoc()函数返回的数组*/
while($result=mysql_fetch_assoc($arr)){                    //循环输出查询结果
?>
      <tr>
      <td height="25"><?php echo $result['name'];?>  </td>      <!--输出新闻标题-->
            <td height="25"><?php echo $result['news'];?> </td>           <!--输出新闻内容-->
     </tr>
<?php
  }                                                        //结束 while 循环
?>
```

运行结果如图 21.14 所示。

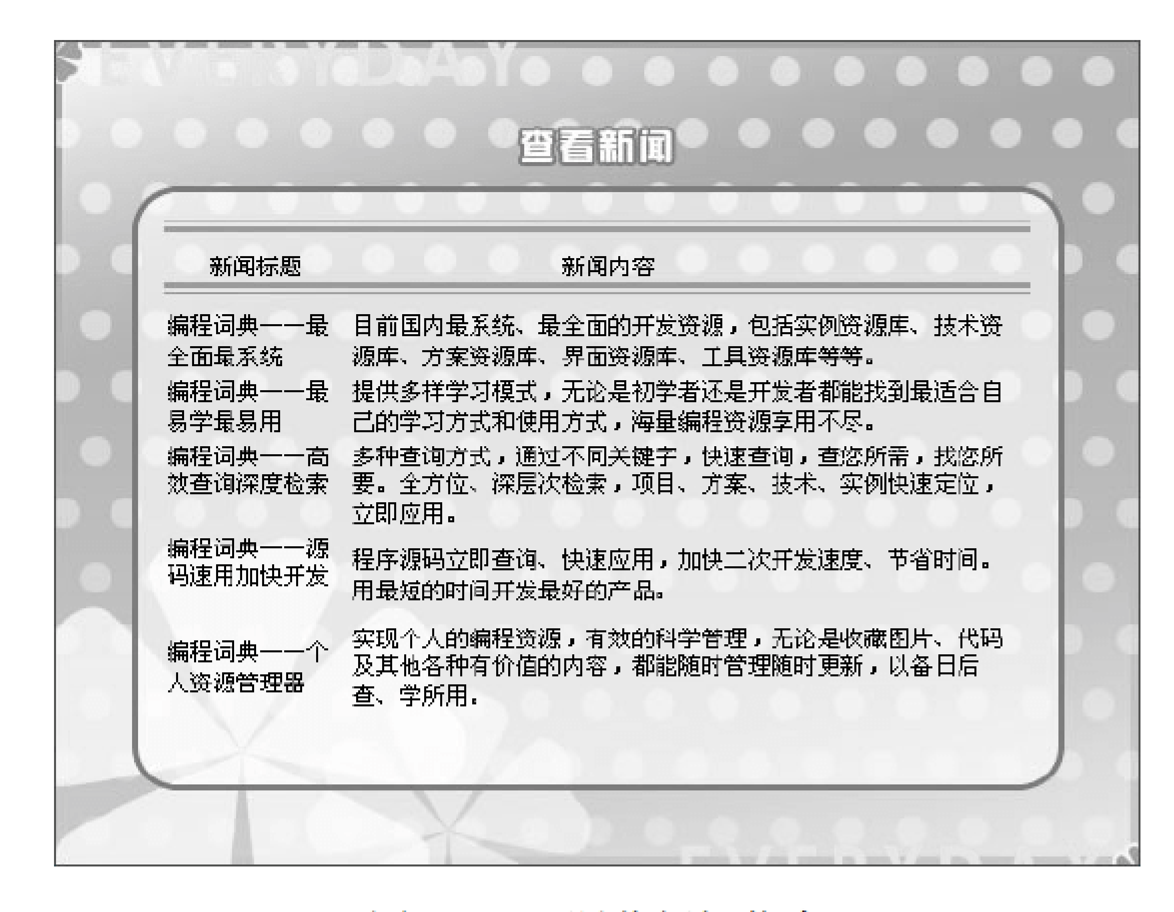


图 21.14　浏览新闻信息

21.4.3　编辑数据库数据

编辑数据主要通过 mysql_query()函数和 update 语句实现。

例 21.12　编辑新闻信息表中的新闻信息，具体步骤如下。**（实例位置：光盘\TM\sl\21\21.12）**

（1）创建数据库连接文件 conn.php，代码如下。

```
<?php
$conn=mysql_connect("localhost","root","root");          //连接数据库服务器
mysql_select_db("db_database21",$conn);                  //连接 db_database21 数据库
mysql_query("set names uft8");                           //设置数据库编码格式
?>
```

（2）创建 index.php 文件，显示所有新闻信息，代码如下。

```
<?php
include("conn.php");                                          //包含 conn.php 文件
$arr=mysql_query("select * from tb_news",$conn);              //查询数据
/*使用 while 语句循环 mysql_fetch_array()函数返回的数组*/
while($result=mysql_fetch_array($arr)){
?>
        <tr>
          <td height="25"><?php echo $result['name'];?><!--输出新闻标题--> </td>
          <td><?php echo $result['news'];?>            <!--输出新闻内容-->              </td>
          <td><label>
            <input type="hidden" name="id" value="<?php echo $result['id'];?>" />
            <div align="center"><a href="update.php?id=<?php echo $result['id'];?>">编辑</a></div>
          </label></td>
          </tr>
<?php
  }                                                           //结束 while 循环
?>
```

（3）创建 update.php 文件，显示要编辑的新闻内容，代码如下。

```
<form id="form1" name="form1" method="post" action="update_ok.php">
<?php
include("conn.php");      //包含 conn.php 文件
$arr=mysql_query("select * from tb_news where id='".$_GET['id']."'",$conn);          //定义查询语句
$select=mysql_fetch_array($arr);                                                     //循环输出查询内容
?>
     <input name="name" type="text" size="40" value="<?php echo $select['name'];?>"/>
    <textarea name="news" cols="40" rows="10"><?php echo $select['news'];?></textarea>
    <input type="submit" name="Submit" value="保存" />
    <input type="hidden" name="id" value="<?php echo $select['id'];?>" />
</form>
```

（4）创建 update_ok.php 文件，完成新闻信息的编辑操作，代码如下。

```
<?php
include("conn.php");                                    //包含 conn.php 文件
if(isset($_POST['id']) and isset($_POST['Submit']) and $_POST['Submit']=="保存"){
    $update=mysql_query("update tb_news set name='".$_POST['name']."',news='".$_POST['news']."' where
id='".$_POST['id']."'",$conn);
    if($update){
        echo  "<script> alert('修改成功!'); window.location.href='index.php'</script>";
    }else{
        echo  "<script> alert('修改失败!'); window.location.href='index.php'</script>";
    }
}
?>
```

运行结果如图 21.15 所示。

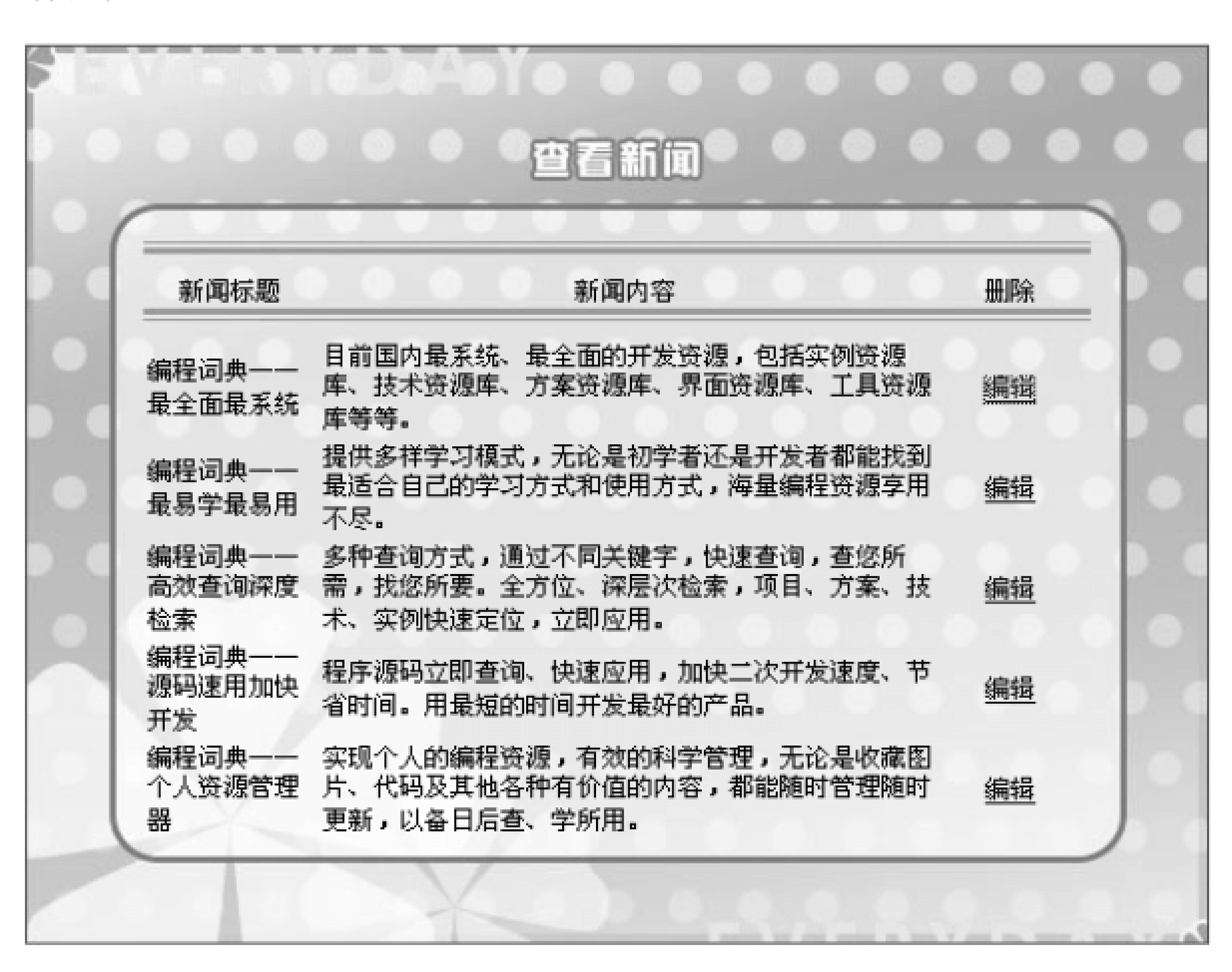


图 21.15　编辑新闻信息

21.4.4　删除数据

数据的删除应用 delete 语句，而在 PHP 中需要通过 mysql_query()函数来执行这个 delete 删除语句，完成 MySQL 数据库中数据的删除操作。

例 21.13　删除新闻信息表中的新闻信息，具体步骤如下。**（实例位置：光盘\TM\sl\21\21.13）**

（1）创建数据库连接文件 conn.php，代码如下。

```
<?php
$conn=mysql_connect("localhost","root","root");                //连接数据库服务器
mysql_select_db("db_database21",$conn);                        //连接数据库 db_database21
mysql_query("set names uft8");                                 //设置数据库编码格式
?>
```

（2）创建 delete.php 文件，显示所有新闻信息，代码如下。

```
<?php
include("conn.php");                                          //包含 conn.php 文件
$arr=mysql_query("select * from tb_news",$conn);              //查询数据
/*使用 while 语句循环 mysql_fetch_array()函数返回的数组*/
while($result=mysql_fetch_array($arr)){
?>
        <tr>
          <td height="25"><?php echo $result['name'];?>       <!--输出新闻标题--> </td>
          <td><?php echo $result['news'];?>                   <!--输出新闻内容-->        </td>
          <td><label>
            <input type="hidden" name="id" value="<?php echo $result['id'];?>" />
             <div align="center"><a href="delete_ok.php?id=<?php echo $result['id'];?>">删除</a></div>
          </label></td>
          </tr>
<?php
  }                                                           //结束 while 循环
?>
```

（3）在 delete_ok.php 文件中，根据超链接传递的 ID 值，完成删除新闻信息操作，代码如下。

```
<?php
include("conn.php");                                                          //包含 conn.php 文件
    $delete=mysql_query("delete from tb_news where id='".$_GET['id']."'",$conn);     //执行删除操作
    if($delete){
        echo   "<script> alert('删除成功!'); window.location.href='delete.php'</script>";
    }else{
        echo   "<script> alert('删除失败!'); window.location.href='delete.php'</script>";
    }
?>
```

执行程序，运行结果如图 21.16 所示。

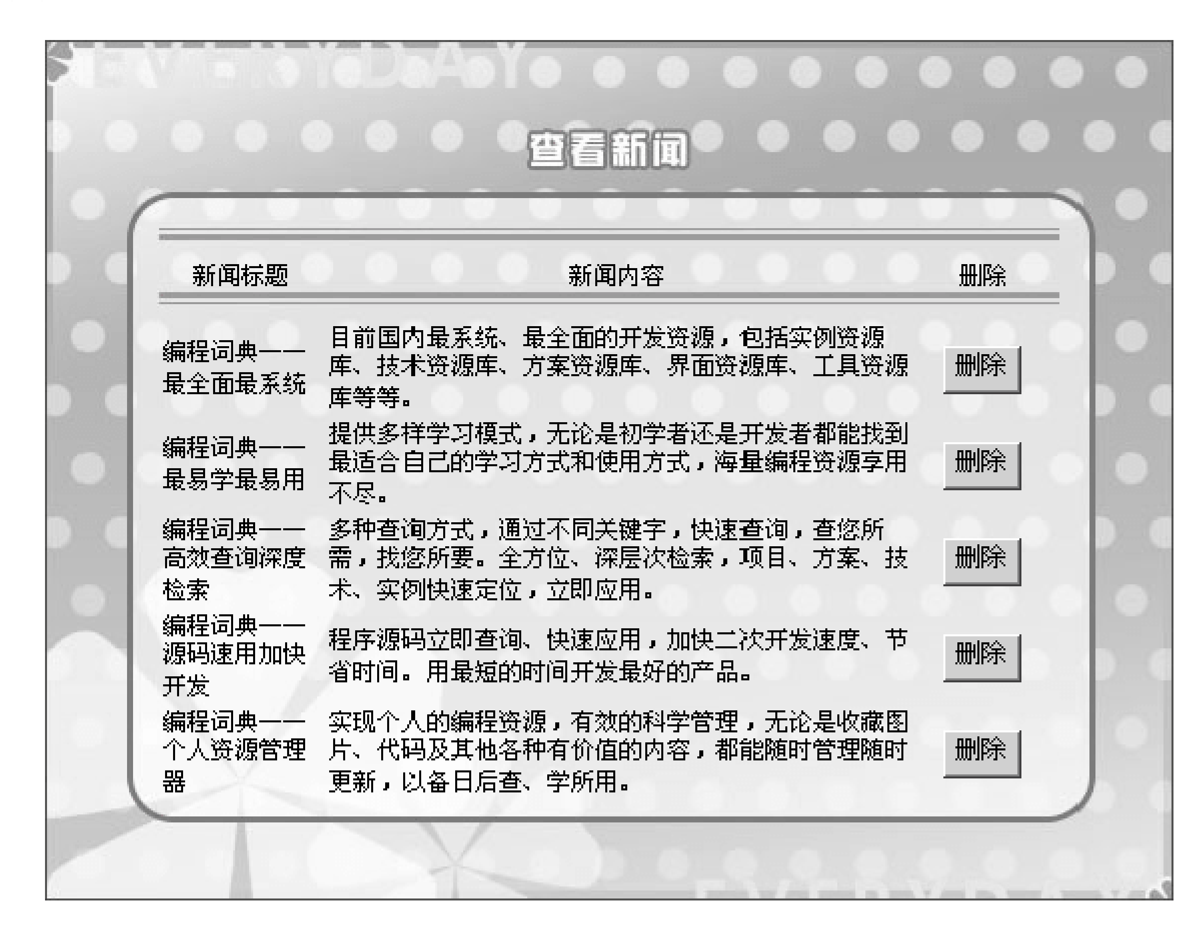


图 21.16　删除新闻信息

21.4.5 批量删除数据

对数据库中的数据进行管理过程中，如果要删除的数据非常多，执行单条删除数据的操作就不适合了，这时应该使用批量删除数据来实现数据库中信息的删除。通过数据的批量删除可以快速删除多条数据，减少操作执行的时间。

例 21.14 批量清理新闻信息表中陈旧的新闻信息，具体步骤如下。**（实例位置：光盘\TM\sl\21\21.14）**

（1）创建数据库连接文件 conn.php，代码如下。

```
<?php
$conn=mysql_connect("localhost","root","root");              //连接数据库服务器
mysql_select_db("db_database21",$conn);                      //连接 db_database21 数据库
mysql_query("set names uft8");                               //设置数据库编码格式
?>
```

（2）创建 pl_delete.php 文件，显示所有新闻信息，代码如下。

```
<?php
include("conn.php");
$arr=mysql_query("select * from tb_news",$conn);             //查询数据
/*使用 while 语句循环 mysql_fetch_array()函数返回的数组*/
while($result=mysql_fetch_array($arr)){
?>
          <tr>
            <td><label>
              <label>
              <input type="checkbox" name="checkbox[]" value="<?php echo $result['id'];?>" />
              </label>
            </label></td>
            <td height="25"><?php echo $result['name'];?><!--输出新闻标题--></td>
            <td><?php echo $result['news'];?><!--输出新闻内容--></td>
          </tr>
<?php
  }                                                          //结束 while 循环
?>
```

（3）创建 pl_delete1.php 页面，完成批量删除操作，代码如下。

```
<?php
include("conn.php");                                         //包含 conn.php 文件
if(isset($_POST['Submit']) and $_POST['Submit']=="删除" and $_POST['checkbox']!=""){//判断是否执行删除操作
    for($i=0;$i<count($_POST['checkbox']);$i++){             //遍历复选框获取到的新闻 id 序号
        $sql=mysql_query("delete from tb_news where id='".$_POST['checkbox'][$i]."'",$conn);//执行删除操作
    }
    if($sql){
```



```
        echo  "<script> alert('删除成功!'); window.location.href='pl_delete.php'</script>";
    }else{
        echo  "<script> alert('删除失败!'); window.location.href='pl_delete.php'</script>";
    }
}else{
    echo  "<script> alert('请选择要删除的内容!'); window.location.href='pl_delete.php'</script>";
}
?>
```

执行程序，运行结果如图 21.17 所示。

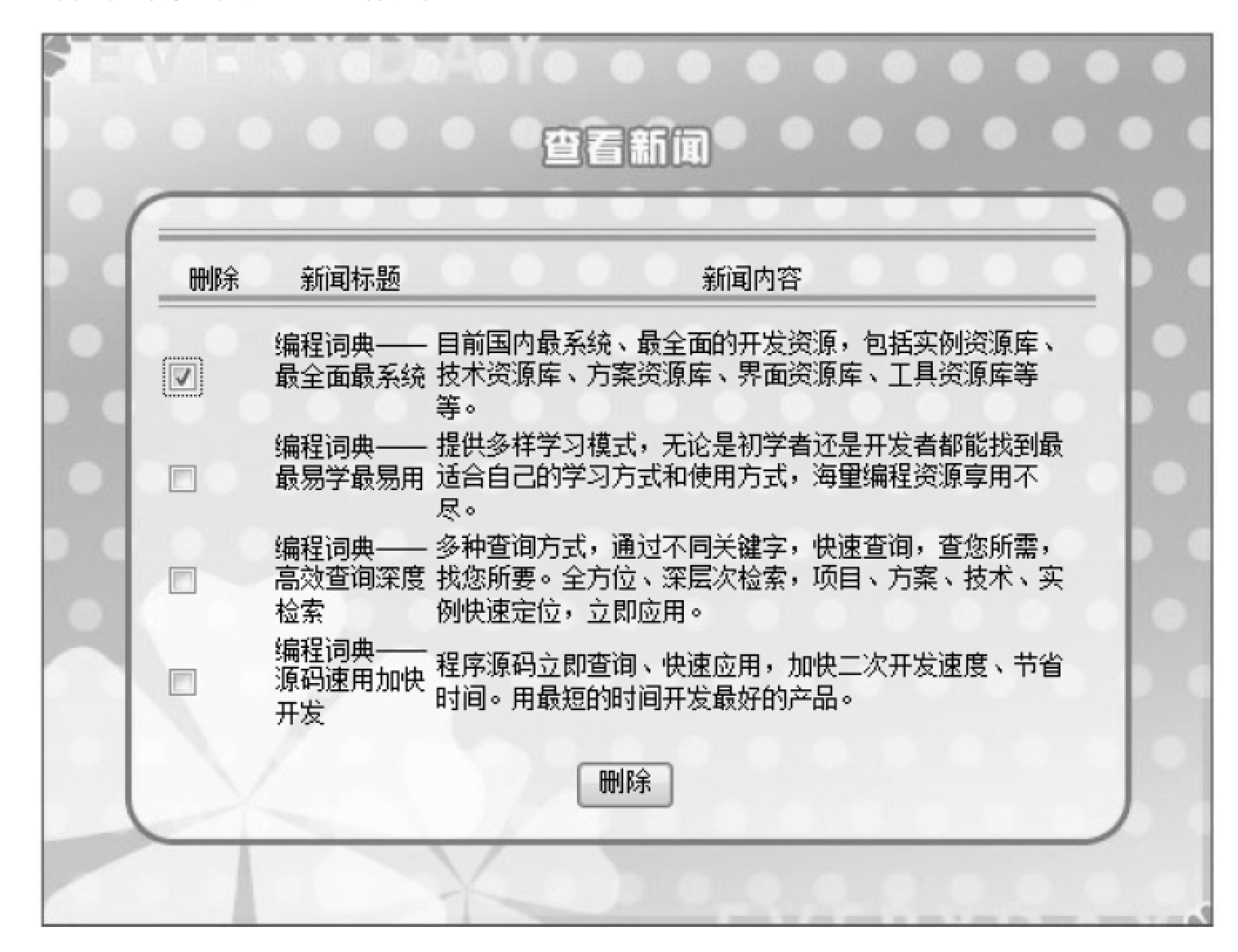


图 21.17　批量删除数据信息

21.5　PHP 操作 MySQL 事务

事务处理机制在程序开发过程中有着非常重要的作用，它可以使整个系统更加安全。例如，在银行处理转账业务时，如果 A 账户中的金额刚被发出，而 B 账户还没来得及接收就发生停电，会给银行和个人带来很大的经济损失。采用事务处理机制，一旦在转账过程中发生意外，则程序将回滚，不做任何处理。

下面讲解使用事务处理技术实现关联表信息的删除方法。

例 21.15　采用事务处理方式，对学生信息表和学生成绩表中的数据进行删除。运行本实例，学生信息表及学生成绩表信息分别如图 21.18 和图 21.19 所示。当删除图 21.18 中的学生信息后，查看学生成绩信息时可以发现，与该学生对应的成绩也被全部删除。代码如下。（**实例位置：光盘\TM\sl\21\21.15**）

学生成绩管理系统

查看学生信息　查看学生成绩

学生姓名	学号	年龄	住址	身份证号	操作
lzh	0312317	18	吉林省长春市	22010245****	删除
lbh	0312316	17	吉林省吉林市	20211032855*****	删除

图 21.18　查看学生信息

学生成绩管理系统

查看学生信息　查看学生成绩

学生姓名	学号	语文	数学	外语
lzh	0312317	85	98	78
lbh	0312316	75	98	86
lbh	0312316	79	95	80
lzh	0312317	78	82	89

图 21.19　查看学生成绩

程序开发步骤如下。

（1）建立数据库及数据表并实现与数据的连接，代码如下。

```
<?php
$conn=new mysqli("localhost","root","root","db_database21");
$conn->query("set names utf8");
?>
```

（2）显示所有学生的基本信息，代码如下。

```
<?php
     include_once("conn.php");
    $sql=$conn->query("select * from tb_stu");
    $info=$sql->fetch_array(MYSQLI_ASSOC);
    if($info==NULL)
     {
       echo "暂无学生信息!";
     }
     else
     {
       do
        {
?>
   <tr>
        <td height="25" bgcolor="#FFFFFF"><div align="center"><?php echo $info[sname];?></div></td>
        <td height="25" bgcolor="#FFFFFF"><div align="center"><?php echo $info[sno];?></div></td>
        <td height="25" bgcolor="#FFFFFF"><div align="center"><?php echo $info[sage];?></div></td>
        <td height="25" bgcolor="#FFFFFF"><div align="center"><?php echo $info[saddress];?></div></td>
```

```
        <td height="25" bgcolor="#FFFFFF"><div align="center"><?php echo $info[ssfzh];?></div></td>
        <td bgcolor="#FFFFFF"><div align="center"><a href="javascript:if(window.confirm('确定删除该学生信息么?')==true){window.location.href='delete.php?id=<?php echo $info[id];?>';}">删除</a></div></td>
      </tr>
 <?php
   }
        while($info=$sql->fetch_array(MYSQLI_ASSOC));
   }
 ?>
```

在实现该模块功能时，可以利用 JavaScript 实现该页与 delete.php 页面的信息传递，代码如下。

```
<a href="javascript:if(window.confirm('确定删除该学生信息嘛')==true)
{window.location.href='delete.php?id=<?php echo $info[id];?>';}">删除
</a>
```

window 对象的 confirm()方法用于弹出一个对话框，提示用户真正删除某学生的所有信息。

（3）利用事务处理机制对学生的基本信息进行删除，代码如下。

```
<?php
$id=$_GET[id];
include_once("conn.php");
$conn->autocommit(false);
if(!$conn->query("delete from tb_sco where id='".$id."'"))
{
 $conn->rollback();
}
if(!$conn->query("delete from tb_stu where id='".$id."'"))
{
 $conn->rollback();
}
$conn->commit();
$conn->autocommit(true);
echo "<script>window.location.href='index.php';</script>";
?>
```

21.6　PHP 操作 MySQL 存储过程

MySQL 5.0 以后的版本开始支持存储过程，存储过程具有一致性、高效性、安全性和体系结构等特点，本节将具体讲解 PHP 是如何操纵 MySQL 存储过程的。

下面介绍如何使用存储过程实现用户登录。

例 21.16　使用 MySQL 存储过程实现用户登录身份的验证。运行本实例，如图 21.20 所示。在图中登录窗口的文本框中输入用户名和密码，然后单击“登录”按钮，如果用户登录的用户名和密码正确，则会将页面定向到如图 21.21 所示的登录成功提示页面。代码如下。（**实例位置：光盘\TM\sl\21\21.16**）



图 21.20　用户登录窗口

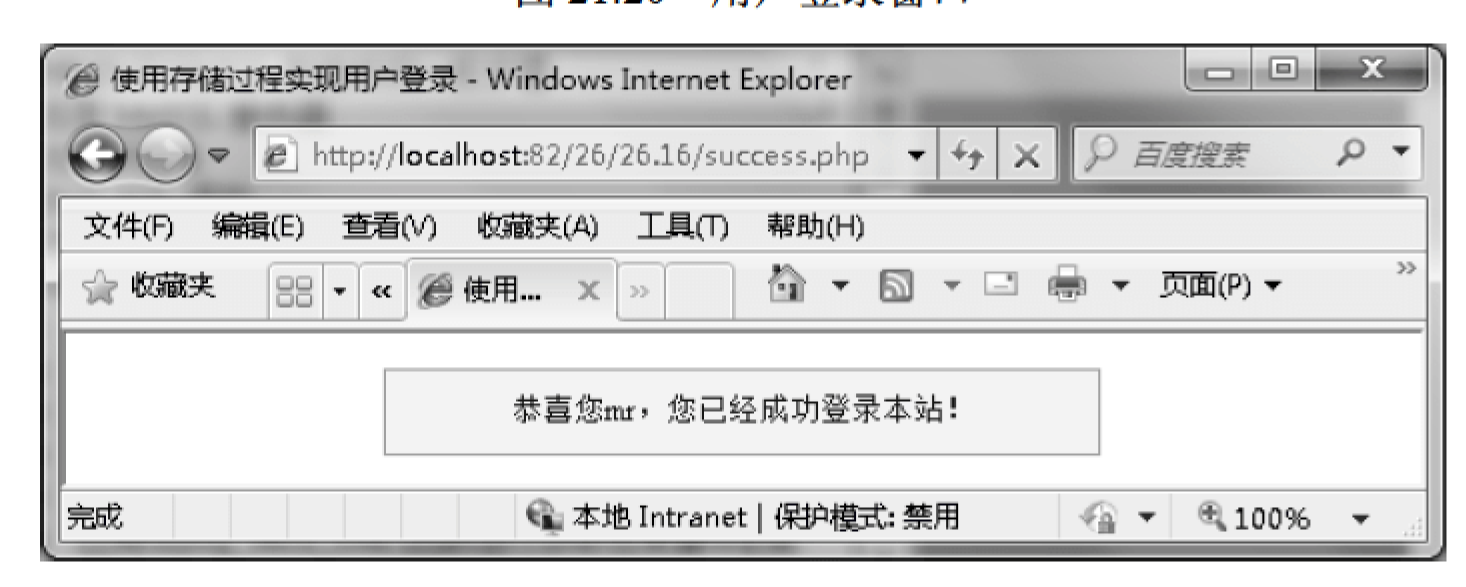


图 21.21　登录成功的提示信息

程序开发步骤如下。

（1）创建存储过程，通过 query 方法调用存储过程，代码如下。

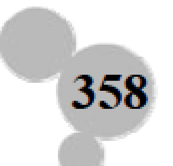
```
delimiter //
create procedure pro_login1(in un varchar(50), in pass varchar(50))
begin
select * from tb_login where username=un and password=pass;
end
//
```

（2）通过 MySQLi 扩展库建立与 MySQL 数据库的连接，并设置数据库字符集为 utf-8，代码如下。

```
<?php
$conn=new mysqli("localhost","root","root","db_database21");
$conn->query("set names utf8");
?>
```

（3）建立用户登录表单，当用户在表单中录入用户名和密码，并单击“提交”按钮后，通过如下代码验证用户的登录信息是否正确。

```
<?php
if(isset($_POST['username']) && trim($_POST['username'])!='')
{
    require_once 'Db.php';
    $username = trim($_POST['username']);
    $password = trim($_POST['password']);
    $sql = $mysqli->query("call pro_login1('".$username."', '".$password."')");
    $info = $sql->fetch_array(MYSQLI_ASSOC);
```



```
    if($info != null){
        $_SESSION['loginUsername'] =  $username;
        echo '<script>window.location.href="success.php";</script>';
    }else {
        echo '<div style="width:300px; height:30px; line-height:30px; border:1px solid #E59B04;
background-color:#FCF2E0; color:#FF0000;">用户名或密码输入有误</div>';
    }
}
?>
```

上述代码中，首先判断$_POST['username']的值是否被设置，则首先使用 require_once 语句包含 conn.php 文件，然后使用 MySQLi 扩展库的 query()方法执行存储过程 pro_login，并使用 query()函数返回的对象调用 fetch_array()函数获得结果集，最后通过判断结果集是否为 null 来判断用户所录入的信息是否正确。

21.7　常见问题与解决方法

在实际 PHP 与 MySQL 结合的应用中，往往会碰到很多问题。这些问题不一定是由于代码的错误造成的，可能是一些环境因素，或者是数据存在的问题。本节将介绍一些常见问题及相应的解决方案。

21.7.1　MySQL 服务器无法连接

错误信息如下所示。

```
Warning:mysql_connect() [function.mysql-connect]: Unkown MySQL server host 'localhost'(11001) in
E:\wamp\www\test.php on line 2
```

出现这条错误信息的原因可能有以下几点。

（1）代码中的 mysql_connect 函数中指定的服务器地址有误。

（2）数据库服务器不可用。

解决方案如下。

（1）检查代码中的服务器地址是否正确。

（2）检查数据库服务器是否已经启动并且可用。

21.7.2　用户无权限访问 MySQL 服务器

错误信息如下。

```
Warning:mysql_connect() [function.mysql-connect]: Access denied for user 'root'@ 'localhost'(using
password:NO) in E:\wamp\www\test.php on line 2
```

出现这条错误信息的原因可能是代码中的 mysql_connect 函数中能够指定的用户名或者密码有误或者在当前服务器上不可用。

解决方案如下。

检查代码中的用户名和密码是否正确。

通过 MySQL 命令行测试是否可以使用该用户名和密码登录 MySQL 数据库服务器。

21.7.3　提示 mysql_connect 函数未定义

错误信息如下。

```
Fatal error : Call to undefined function mysql_connect() in E:\wamp\www\test.php on line 2
```

出现这条错误信息的原因可能是在 php.ini 文件中没有配置 MySQL 的扩展库。

解决方案如下。

编辑 php.ini 文件，定位到如下位置，去掉此项前面的分号，保存后重新启动 Apache 服务器。

```
;extension=php_mysql.dll
```

21.7.4　SQL 语句出错或没有返回正确的结果

这种情况经常在使用动态 SQL 语句时出现，以下代码就存在一个错误。

```
<?php
    mysql_connect("localhost", "root", "root") or die;
    mysql_select_db("db_database21 ");
    $sql="select * from $table";
    $result=mysql_query($sql);
    print_r(mysql_fetch_row($result));
?>
```

上述代码中错误地使用了一个没有赋值的变量$table 作为操作的数据表名称，结果返回如下错误信息。

```
Warning: mysql_fetch_row(): supplied argument is not a valid MySQL result resource in E:\wamp\www\index.php on line 6
```

解决方案：使用 print 或者 echo 函数输出 SQL 语句来检查错误。例如，对上述代码进行修改，通过 echo 语句直接输出$sql 的值，查看这个 SQL 语句是否正确，其代码如下。

```
<?php
    mysql_connect("localhost", "root", "root") or die;          //连接数据库
    mysql_select_db("db_database21 ");                          //选择数据库
    $sql="select * from $table";                                //定义 SQL 语句
    echo $sql;                                                  //输出 SQL 语句
    $result=mysql_query($sql);                                  //执行 SQL 语句
```



```
    print_r(mysql_fetch_row($result));                    //输出执行结果
?>
```

如此，从运行结果就可以看出 SQL 语句中的错误了。

21.7.5　数据库乱码问题

在获取数据库中的数据时，中文字符串的输出出现乱码。

问题分析：输出数据库中的数据之所以会出现乱码，是因为在获取数据库中的数据时，数据本身所使用的编码格式与当前页面的编码格式不符，从而导致输出数据乱码。

解决方案：在与 MySQL 服务器和指定数据库建立连接后，应用 mysql_query()函数设置数据库中字符的编码格式，使其与页面中的编码格式一致。

```
<?php
$conn=mysql_connect("localhost","root","root");       //连接数据库服务器
mysql_select_db("db_database21",$conn);               //连接数据库 db_database21
mysql_query("set names uft8");                        //设置数据库编码格式
?>
```

上述通过 mysql_query()函数设置的编码格式是 uft8，同样还可以设置其他编码格式，唯一的一个条件就是要与数据库中的编码格式相匹配。

这就是解决数据库中中文输出乱码的方法，应用 mysql_query 函数设置数据库的编码格式，使其与页面中编码格式保持一致，也就不会出现乱码的问题。

21.7.6　应用 MYSQL_ERROR()语句输出错误信息

在执行 MySQL 语句时产生的错误是很难发现的，因为在 PHP 脚本中执行一个 MySQL 的添加、查询、删除语句时，如果是 MySQL 语句本身的错误，程序中不会输出任何的信息，除非对 MySQL 语句的执行进行判断，成功输出什么，失败输出什么。

解决方案：为了查找出 MySQL 语句执行中的错误，可以通过 mysql_error()语句来对 SQL 语句进行判断，如果存在错误则返回错误信息，否则没有输出，该语句的应用被放置于 mysql_query()函数之后。

例如，在下面的代码中，在通过 mysql_query()函数执行查询语句之后，应用 mysql_error()函数获取 SQL 语句中的错误。

```
<?php
        $sql="select * from tb_new";                  //定义查询语句
        $query=mysql_query($sql,$conn);               //执行查询操作
        echo mysql_error();                           //获取 SQL 语句中的错误
        while($myrow=mysql_fetch_array($query)){      //循环输出查询结果
?>
```

此方法不仅对查询语句的执行有效，而且对添加、更新和删除语句都适用，是一个查找 SQL 语句本身错误的好方法。

21.8 小　　结

本章首先对 PHP 语言进行简单的介绍，让读者先对 PHP 语言有一个基本的了解，然后介绍了 PHP 访问 MySQL 数据库的一般流程，并且详细介绍了该流程每一步骤的具体实现方法，接下来再通过实例更深层次地介绍了 PHP 如何实现对 MySQL 数据库进行增、删、改、查操作，最后介绍了如何在 PHP 中操作事务和存储过程，以及 PHP 管理 MySQL 数据库时经常遇到的问题的解决方法。通过本章的学习，读者能够掌握 PHP 操作 MySQL 数据库的一般流程，掌握常用 MySQL 函数的使用方法，并能够具备独立完成基本数据库程序的能力。希望本章能够起到抛砖引玉的作用，能够帮助读者在此基础上更深层次地学习 PHP 操作 MySQL 数据库的相关技术，并进一步学习使用面向对象的方式操作 MySQL 数据库的方法。

21.9 实践与练习

1．编写一个实例，实现以分页的方式显示员工信息，并通过输入页码跳转到指定的页。**（答案位置：光盘\TM\sl\21\21.17）**

2．编写一个实例，实现查询图书信息表中的前 3 条记录。**（答案位置：光盘\TM\sl\21\21.18）**

3．编写一个实例，实现对图书信息表中的图书信息按图书序号进行降序排列。**（答案位置：光盘\TM\sl\21\21.19）**



项目实战

本篇分别使用 PHP 和 Java 两种语言，结合 MySQL 实现了两个大型的、完整的管理系统，通过这两个项目，运用软件工程的设计思想，让读者学习如何进行软件项目的实践开发。书中按照编写系统分析→系统设计→数据库与数据表设计→公共模块设计→创建项目→实现项目→项目总结的过程进行介绍，带领读者一步一步亲身体验开发项目的全过程。

第22章 Apache+PHP+MySQL 实现网上社区

所谓网上社区是指包括BBS/论坛、聊天室、博客等形式在内的网上交流空间，同一主题的网上社区集中了具有共同兴趣的访问者，由于有众多用户的参与，因此具备了交流的功能，成为一个营销场所。网上社区有各种不同的表现形式和规模，有个人创办的社区，功能和界面追求时尚、个性突出；有大型的商业性质社区，以盈利为目的，分类多元化，适合不同类型的网民。

本章开发的BCTY365网上社区主要面向程序开发人员，集论坛、留言板、软件下载、升级下载、技术支持和在线购物等功能于一身，既是一个程序开发者交流的平台，更是一个网络营销的场所。

通过阅读本章，读者可以：

- 了解如何做需求分析和系统设计
- 掌握在Linux操作系统下搭建PHP开发环境
- 掌握如何设计和创建数据库、数据表
- 掌握如何设计公共模块
- 掌握在线支付技术
- 掌握在线论坛功能的实现方法
- 掌握软件上传和下载功能的实现方法
- 掌握在Linux操作系统下发布网站的方法

22.1　开发背景

随着市场竞争的日益激烈，企业的生存和发展之路更加艰难。要想使企业保持旺盛的生命力，企业必须要跟上时代发展的脚步，不断为企业注入新的活力。某科技公司为适应市场的需求，增加公司在互联网上的影响力，将开发一个网上社区系统，为广大的编程爱好者提供一个交流的平台，并且以此来推广该公司的软件产品。

22.2　系统分析

当一个开发项目被确立时，首先要做的就是需求分析、可行性分析，然后编写项目计划书，以使项目开发人员了解和掌握网站的前期策划和网站开发流程。

22.2.1　需求分析

在开发网上社区之前，首先要明确所要开发的社区属于什么类型，是个人的社区系统，还是商业化的社区系统；并且要知道开发的社区是面向什么样的人群，是普通网民，还是专业的技术人员，或者是其他的特殊群体。针对不同的人群，社区应该具有不同的特点。当明确了这些，项目开发的思路就清晰了，然后再对网站上一些相关的社区进行考察、分析，从中吸取经验，并结合企业的要求以及实际的市场调查结果，提出一个合理的网上社区网站功能架构。本网站需求如下。

（1）网站设计页面要求整洁、美观大方，能够展示企业形象。
（2）网站页面具有 banner 广告，树立企业良好的口碑宣传。
（3）设计主要从编程者的角度考虑，为编程者解决在开发中出现的问题。
（4）展示出企业全力推出的软件产品和提供的免费软件，以此吸引浏览者。
（5）提供一个良好的网上购物的操作平台。
（6）提供技术支持，解决编程过程中常见的问题。
（7）提供一个讨论和研究问题的平台。
（8）做到让广大浏览者关注企业的动态。
（9）为客户提供反馈信息的平台，能够做到及时与客户进行沟通。
（10）完善的后台管理系统。

22.2.2　可行性分析

可行性分析是世界上普遍采用的一种研究工程项目是否可行的科学。其通过各种有效的方法，对工程项目进行分析，从技术、经济、市场等方面加以评价，最终给投资决策者提供是否选择该项目进

行开发的依据。

BCTY365 网上社区项目开发的可行性分析主要从以下两个方面考虑。

1. 经济可行性分析

企业为扩大公司的影响力，推出软件产品，采用网上社区的形式在网络上进行推广，不但可以汇聚更多的人气，而且可以让更多的人了解该企业，从而达到推广企业软件产品的目的，最终为企业带来更大的收益。更重要的一点是采取该方法的成本相对其他的电视广告或者人力宣传的成本要低得多，虽然周期很长，但是却能够取得长期的收益。

2. 技术可行性分析

网上社区系统的开发采用的是 Apache+PHP+phpMyadmin+MySQL 5.6，开发软件都是免费的，直接可以从网上下载，无须支付任何费用。要完成 BCTY365 网上社区系统的开发，必须能够配置 PHP 程序开发的环境，掌握在线支付、购物车和在线论坛技术。

22.3 系统设计

22.3.1 系统目标

根据对目前网络上各种社区的分析和研究，结合本项目的实际需求，在设计时应该达到以下目标。

（1）界面设计美观大方、方便、快捷、操作灵活，树立企业形象。

（2）功能完善、结构清晰。

（3）重点突出企业的软件产品。

（4）及时更新网站公告。

（5）及时查阅和回复客户反馈信息。

（6）为用户提供沟通和交流的平台。

（7）购物车模块的设计结构合理、流程清晰。

（8）购物结算功能设计符合逻辑，计算准确。

（9）订单处理功能的设计及时、准确、安全。

（10）网上支付功能的设计处理好与网上银行之间数据的传递，确保安全、可靠。

（11）具备完善的后台管理功能，能够及时、准确地对网站进行维护和更新。

（12）系统运行稳定，具备良好的防范措施。

22.3.2 系统功能结构

结合需求分析和系统目标中的内容，BCTY365 网上社区系统的功能结构已经设计完成。为了使读者能够更清楚地了解网站的结构，下面给出 BCTY365 网上社区前台和后台功能模块结构图。



BCTY365 网上社区前台管理系统的功能设计如图 22.1 所示。

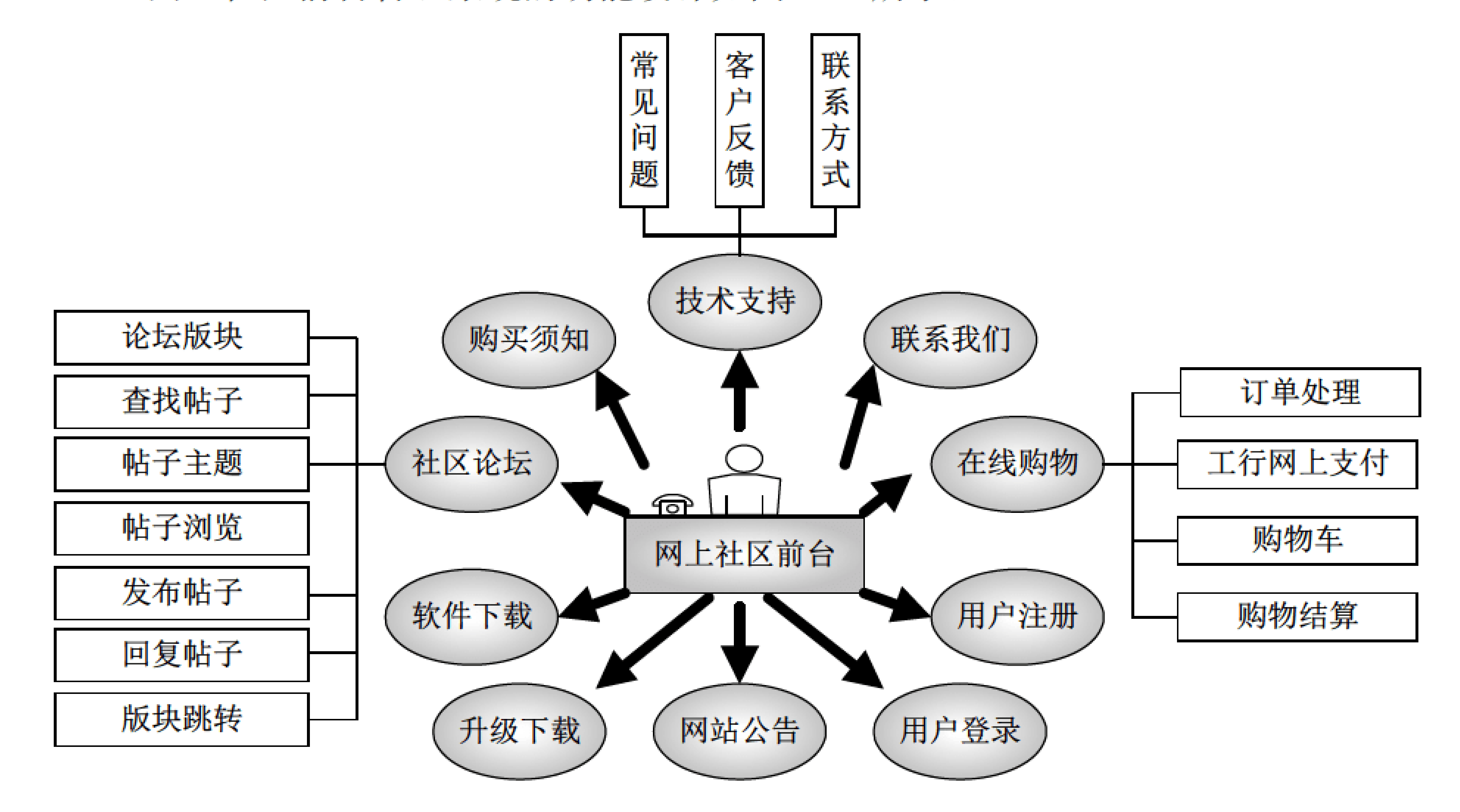


图 22.1　网上社区前台功能模块结构图

BCTY365 网上社区后台管理系统的功能设计如图 22.2 所示。

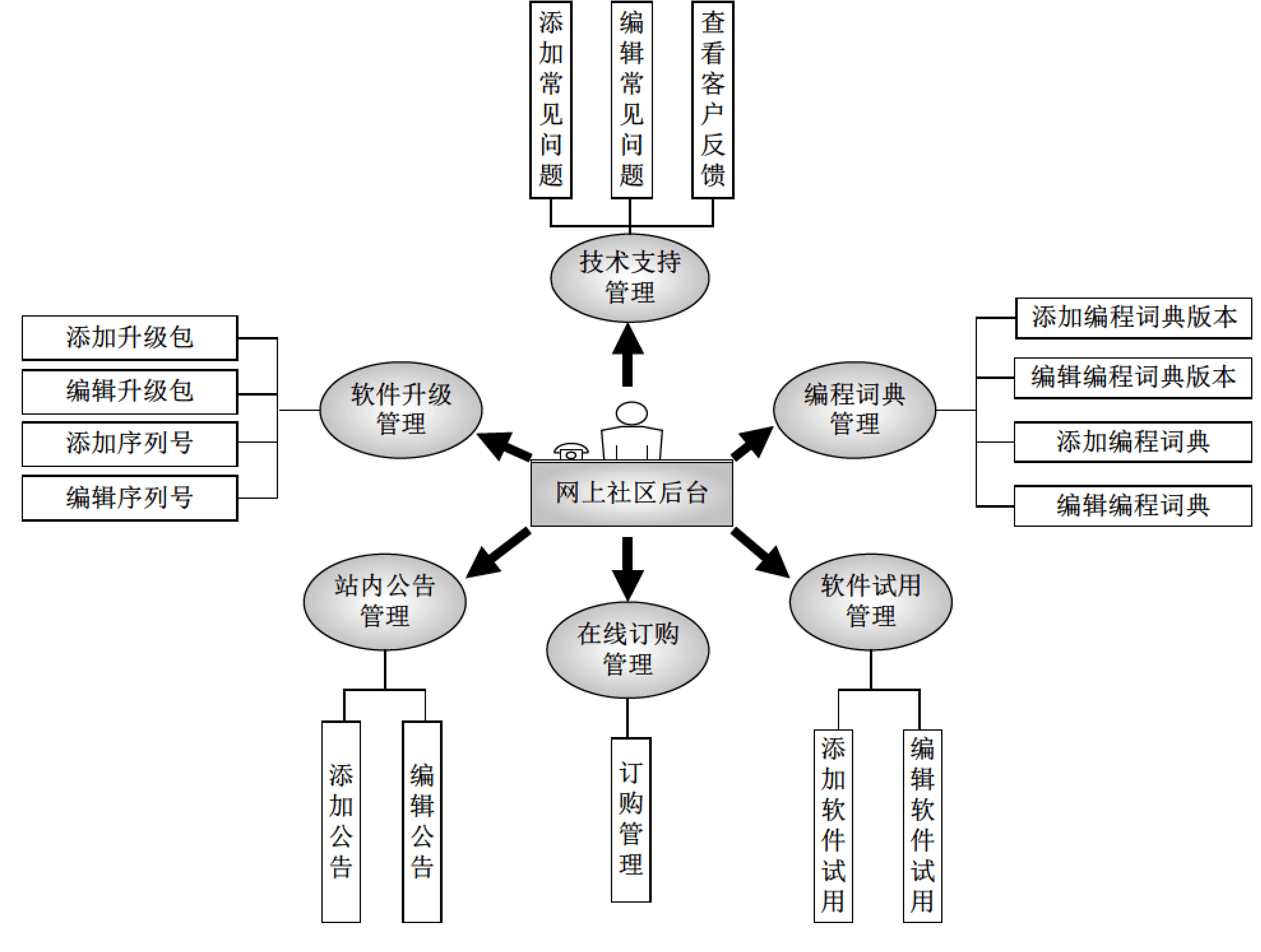


图 22.2　网上社区后台功能模块结构图

22.3.3　系统预览

BCTY365 网上社区系统由多个程序页面组成，下面给出几个典型页面，其他页面参见光盘中的源程序。

前台首页如图 22.3 所示，该页面用于展示本系统的功能模块，突出企业的形象，推广企业的软件产品。后台首页如图 22.4 所示，该页面用于实现对编程词典、技术支持、软件升级、软件试用等内容的管理。





图 22.3　前台首页（光盘\TM\sl\22\bcty365\index.php）　图 22.4　后台首页（光盘\TM\sl\22\bcty365\admin\default.php）

在线订购模块的页面效果如图 22.5 所示，该页面主要用于展示本企业在线推出的软件产品，实现对产品的在线购买功能。软件下载模块的页面效果如图 22.6 所示，该页面主要用于展示本企业提供的免费软件，并且提供下载链接。



图 22.5　在线订购（光盘\TM\sl\22\bcty365\ morebccd.php）　图 22.6　软件下载（光盘\TM\sl\22\bcty365\rjxz.php）

社区论坛模块的页面效果如图 22.7 所示，该页面主要用于展示论坛中的各大版块，并且提供超链接跳转到对应的版块。后台登录页面效果如图 22.8 所示，该页面主要实现后台管理员登录。

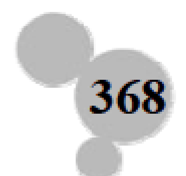




图 22.7　社区论坛（光盘\TM\sl\22\bcty365\bbs_index.php）



图 22.8　后台登录（光盘\TM\sl\22\bcty365\admin\index.php）

22.3.4　开发环境

在开发 BCTY365 网上社区时，该项目使用的软件开发环境如下。

1．服务器端

操作系统：Windows 7/Linux（推荐）。
服务器：Apache 2.2.11。
PHP 软件：PHP 5.2.6。
数据库：MySQL 5.6.20。
MySQL 图形化管理软件：PhpMyAdmin-4.2.8。
开发工具：Dreamweaver 8。
浏览器：IE 6.0 及以上版本。
分辨率：最佳效果 1024×768 像素。

2．客户端

浏览器：IE 6.0 及以上版本。
分辨率：最佳效果 1024×768 像素。

22.3.5　文件夹组织结构

在进行网站开发之前，要对网站的整体文件夹组织架构进行规划。对网站中使用的文件进行合理的分类，分别放置于不同的文件夹下。通过对文件夹组织架构的规划，可以确保网站文件目录明确、条理清晰，同样也便于网站后期的更新和维护。本案例的文件夹组织架构规划如图 22.9 所示。

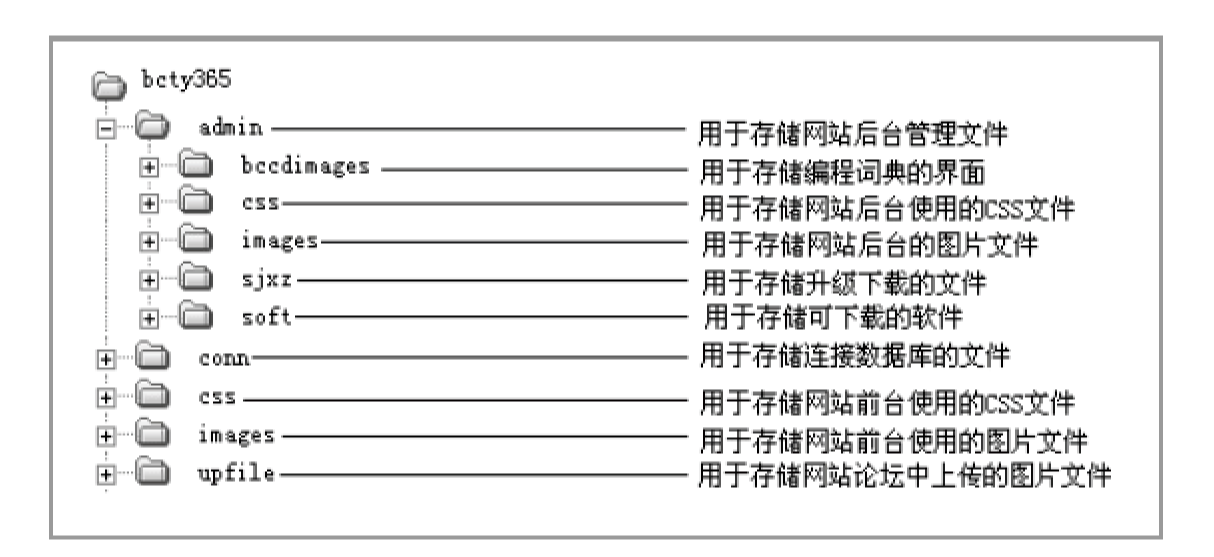


图 22.9　文件夹组织结构

22.4　在 Linux 操作系统下搭建 PHP 开发环境

Red Hat Linux 9 是 Linux 众多版本中比较大众化的一版。在安装系统时，如果选择完全安装或者选择 Apache、MySQL、PHP 的安装包，则三者将被安装到系统中，用户只需将 Apache 和 MySQL 服务启动就可以使用二者，非常方便，但是 Apache 和 MySQL 的版本可能不是很理想。为了能够创建一个良好的 PHP 开发环境，这里将详细介绍自行在 Linux 下安装和配置 Apache 2+MySQL 5.0+PHP 5 的方法。

首先应该到相关官方网站下载三者的安装包。

（1）httpd-2.0.58.tar.gz 或更高版本（http://httpd.apache.org/）。

（2）mysql-standard-5.0.0-alpha-pc-linux-i686.tar.gz 或更高版本（http://www.oracle.com/index.html）。

（3）php-5.0.0.tar.gz 或更高版本（http://www.php.net/）。

libxml2-2.5.10.tar.gz 或更高版本（如果读者系统中的 libxml2 的版本已经等于或高于该版本可以不必下载该安装包）。

22.4.1　Linux 下 Apache 的安装配置

首先将下载的 httpd 安装包复制到适当的位置，例如/usr/local/work 下（如果目录不存在，可以建立该目录）。打开 Red Hat Linux 9 的主菜单，选择“系统工具”，在弹出的菜单中选择“终端”命令，将打开如图 22.10 所示的终端窗口。

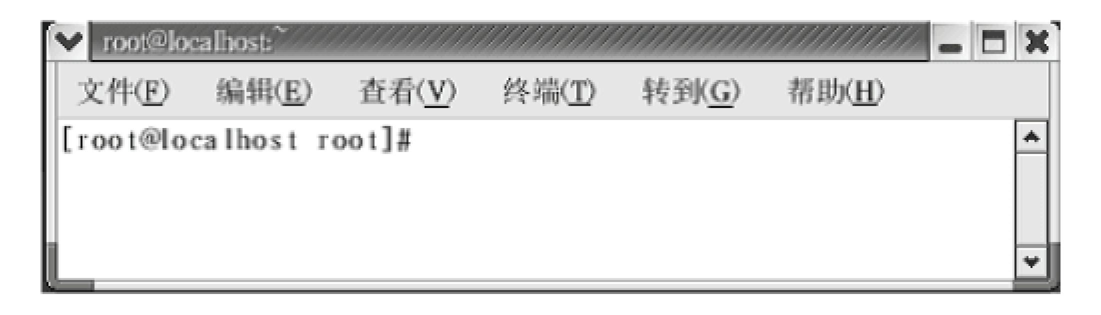


图 22.10　Red Hat Linux 9 的终端命令窗口



Linux 下 Apache、MySQL 及 PHP 的安装都是在如图 22.10 所示的终端命令窗口通过命令方式实现的。

在该窗口中输入如下命令进入 work 目录。

```
cd /usr/local/work
```

在 work 目录中输入如下命令解压 httpd-2.0.58.tar.gz。

```
tar xfz httpd-2.0.58.tar.gz
```

进入解压后的目录 httpd-2.0.58。

```
cd httpd-2.0.58
```

建立 makefile，并将 Apache2 安装到/usr/local/apache2 目录下。

```
./configure –prefix=/usr/local/apache –enable-module=so
```

开始编译。

```
make
```

开始安装到设置的目录去。

```
make install
```

至此，Apache 2 的安装工作完成，可以在每次启动系统时通过如下命令启动或重新启动 Apache 2 服务。

```
/usr/local/apache2/bin/apachectl start
/usr/local/apache2/bin/apachectl restart
```

打开浏览器，在地址栏中输入 http://127.0.0.1 或者 http://localhost，按 Enter 键，如果出现如图 22.11 所示的页面，则说明 Apache 2 安装成功。

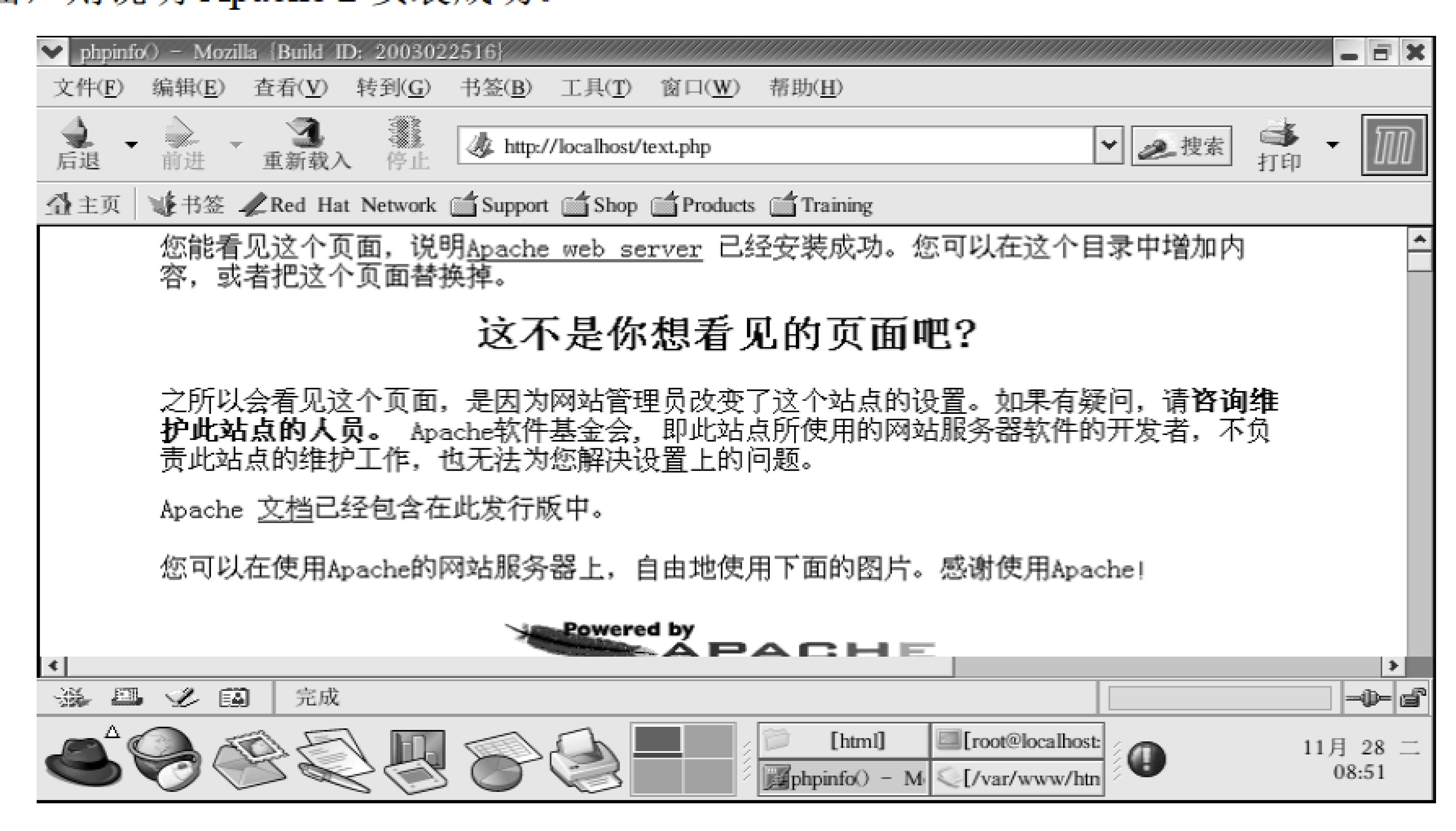


图 22.11　测试 Apache 服务器

22.4.2 Linux 下 MySQL 的安装配置

将 mysql-standard-5.0.0-alpha-pc-linux-i686.tar.gz 复制到/usr/local/work 目录下，建立 MySQL 账号。

```
groupadd mysql
```

在组群中加入 MySQL。

```
useradd –g mysql mysql
```

进入 local 目录。

```
cd /usr/local
```

将 mysql-standard-5.0.0-alpha-pc-linux-i686.tar.gz 解压到该目录。

```
tar xfz /usr/local/work/mysql-standard-5.0.0-alpha-pc-linux-i686.tar.gz
```

考虑到 MySQL 数据库升级的需要，所以通常以链接的方式建立/usr/local/mysql 目录。

```
ln –s mysql-standard-5.0.0-alpha-pc-linux-i686.tar.gz mysql
```

进入 MySQL 目录。

```
cd mysql
```

在/usr/local/mysql/data 中建立 MySQL 的数据库。

```
scripts/mysql_install_db –user=mysql
```

修改文件权限。

```
chown –R root .
chown –R mysql data
chgrp –R mysql .
```

到此 MySQL 安装成功。可以通过在终端中输入如下命令启动 MySQL 服务。

```
/usr/local/mysql/bin/mysqld_safe –user=mysql &
```

启动 MySQL 后输入如下命令查看安装结果。

```
/usr/local/mysql/bin/mysql –uroot
```

如果终端窗口中出现如图 22.12 所示的提示，则说明 MySQL 安装成功。

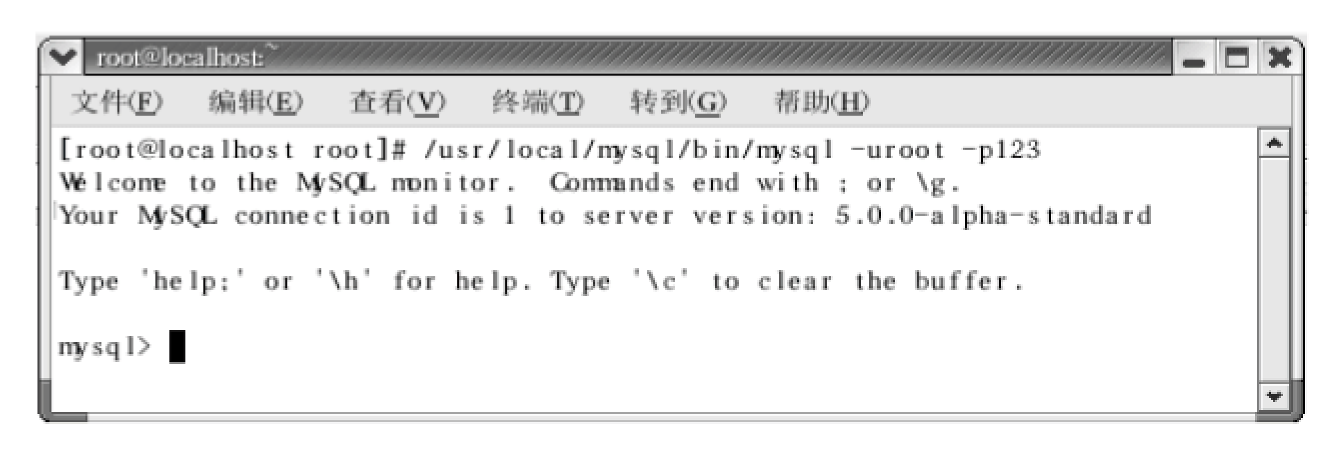


图 22.12 测试 MySQL



22.4.3　Linux 下 PHP 的安装配置

首先查看系统中 libxml2 的版本号，如果 libxml2 的版本号小于 2.5.10，则需要安装 libxml2-2.5.10.tar.gz 或更高版本，因为 PHP 5 必须在 libxml2 的版本大于 2.5.10 的前提下才能够安装。

将 libxml2-2.5.10.tar.gz 复制到/usr/local/work 目录下，并进入该目录。

```
cd /usr/local/work
```

解压 libxml2-2.5.10.tar.gz。

```
tar xfz libxml2-2.5.10.tar.gz
```

进入该目录。

```
cd libxml2-2.10
```

建立 makefie 并将 libxml2 安装到/usr/local/libxml2 下。

```
./configure –prefix=/usr/local/libxml2
```

开始编译。

```
make
```

开始安装到设置的目录去。

```
make install
```

至此 libxml2 安装成功。

将 php-5.0.0.tar.gz 复制到/usr/local/work 目录下，并进入该目录。

```
cd /usr/local/work
```

解压 php-5.0.0.tar.gz。

```
tar xfz php-5.0.0.tar.gz
```

进入 php-5.0.0 目录。

```
cd php-5.0.0
```

建立 makefile。

```
./configure –with-apxs2=/usr/local/apache2/bin/apxs \
--with-mysql=/usr/local/mysql \
--with-libxml-dir=/usr/local/libxml2
```

开始编译。

```
make
```

开始安装。

```
make install
```

复制 php.ini-dist 或 php.ini-recommended 到/usr/local/lib 目录，并命名为 php.ini。

```
cp php.ini-dist /usr/local/lib/php.ini
```

更改 httpd.conf 文件相关设置，该文件位于/usr/local/apache2/conf 中。找到该文件中的如下指令行。

```
AddType application/x-gzip .gz .tgz
```

在该指令后添加如下指令。

```
AddType application/x-httpd-php .php .phtml
```

重新启动 Apache，并在 Apache 主目录下建立文件 test.php。

```
<?php
phpinfo();
?>
```

在浏览器地址栏中输入 http://127.0.0.1/test.php，按 Enter 键，如果出现如图 22.13 所示的页面，则 PHP 安装成功。

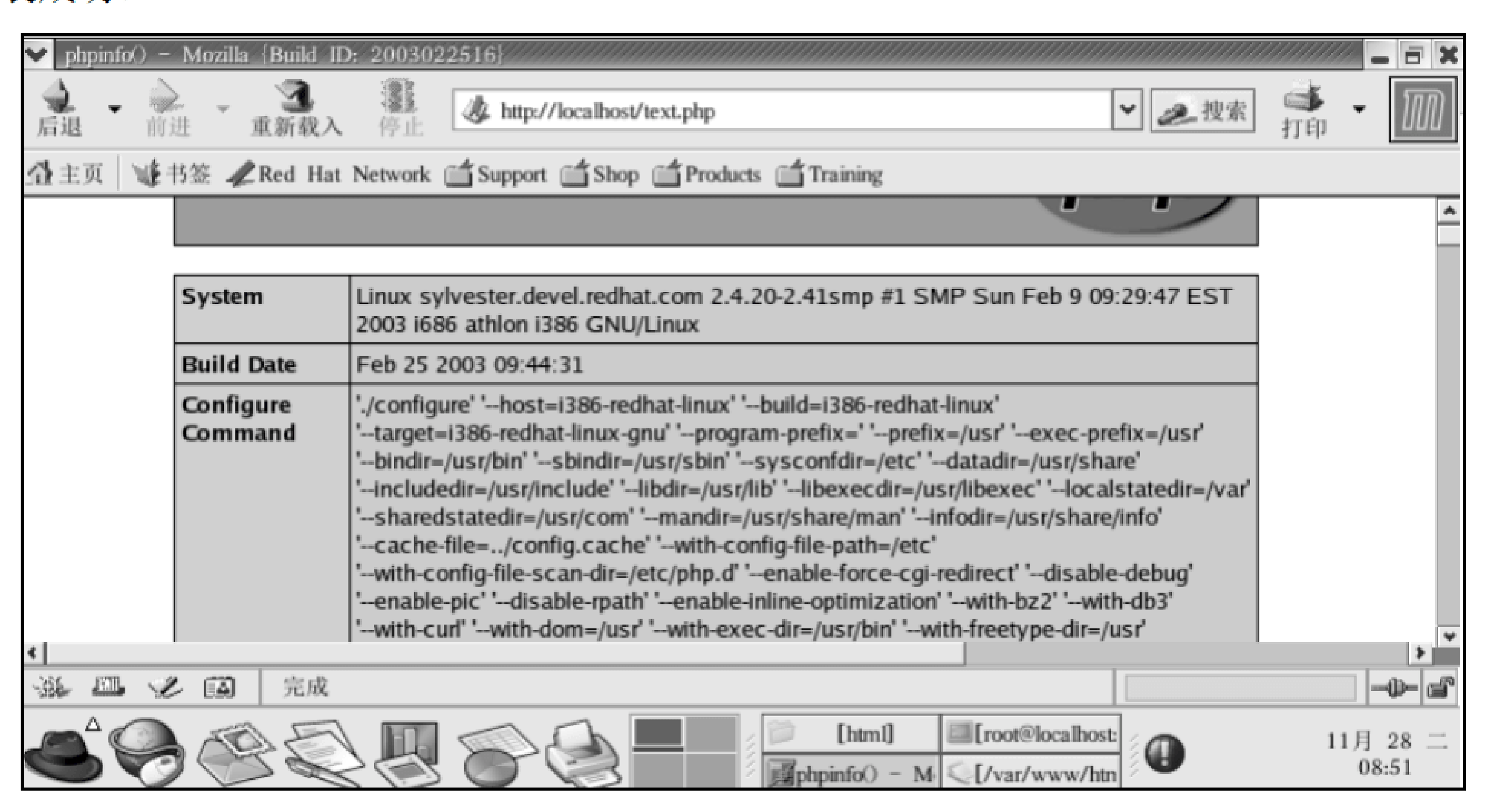


图 22.13　PHP 测试



说明

Apache2 默认主目录为/usr/local/apache2/htdocs。

技巧

安装文件的路径要遵循一定的客观原则，为了避免在 Windows 和 Linux 间移植程序时带来的不便，选择 D:\usr\local\php 的目录时要和在 Linux 下的安装目录相匹配。建议最好不要选择中间有空格的目录，如 E:\program Files\PHP，这样做会导致一些未知错误或崩溃发生。

22.5　数据库设计

开发一个功能完善的网上社区离不开数据库的支持，只有拥有了强大的数据库，网上社区才能够存储大量的数据信息，实现更多、更好的功能来吸引更多的社区成员。本节将对 BCTY365 网上社区数据库的设计进行详细介绍。

22.5.1　数据库分析

BCTY365 网上社区是一个中型的面向软件开发者的程序，考虑到开发的成本、搭配的合理性以及操作的灵活性等，使用了 MySQL 数据库。从成本考虑 MySQL 数据库是完全免费的，可以在网上免费下载；从匹配的角度讲 PHP 与 MySQL 数据库一直是公认的最佳搭档；MySQL 数据库不但可以在命令模式下进行操作，而且还配备了一些比较流行的图形化管理工具，如 phpMyAdmin 等，可以轻松地对 MySQL 数据库中的数据进行操作。

22.5.2　数据库概念设计

根据上述各节对 BCTY365 网上社区系统做的需求分析和系统设计，规划出 BCTY365 网上社区的实体关系 E-R 图。其中包括注册用户实体、发帖信息实体、回帖信息实体、订单信息实体和编程词典信息实体，其他还有一些辅助的实体，是对上述实体中内容的补充。由于涉及的实体较多，这里只对注册用户实体、发帖信息实体和订单信息实体的 E-R 图进行介绍。

1．注册用户实体

注册用户实体用于存储用户注册信息，包括编号、用户名、真实姓名、密码、邮箱、性别、电话、QQ 号码、家庭地址、访问次数、注册时间、最后一次登录时间、IP 地址、邮政编码、用户类型、密码提示问题、密码答案、真实密码、表情图、发帖次数属性。注册用户实体的 E-R 图如图 22.14 所示。

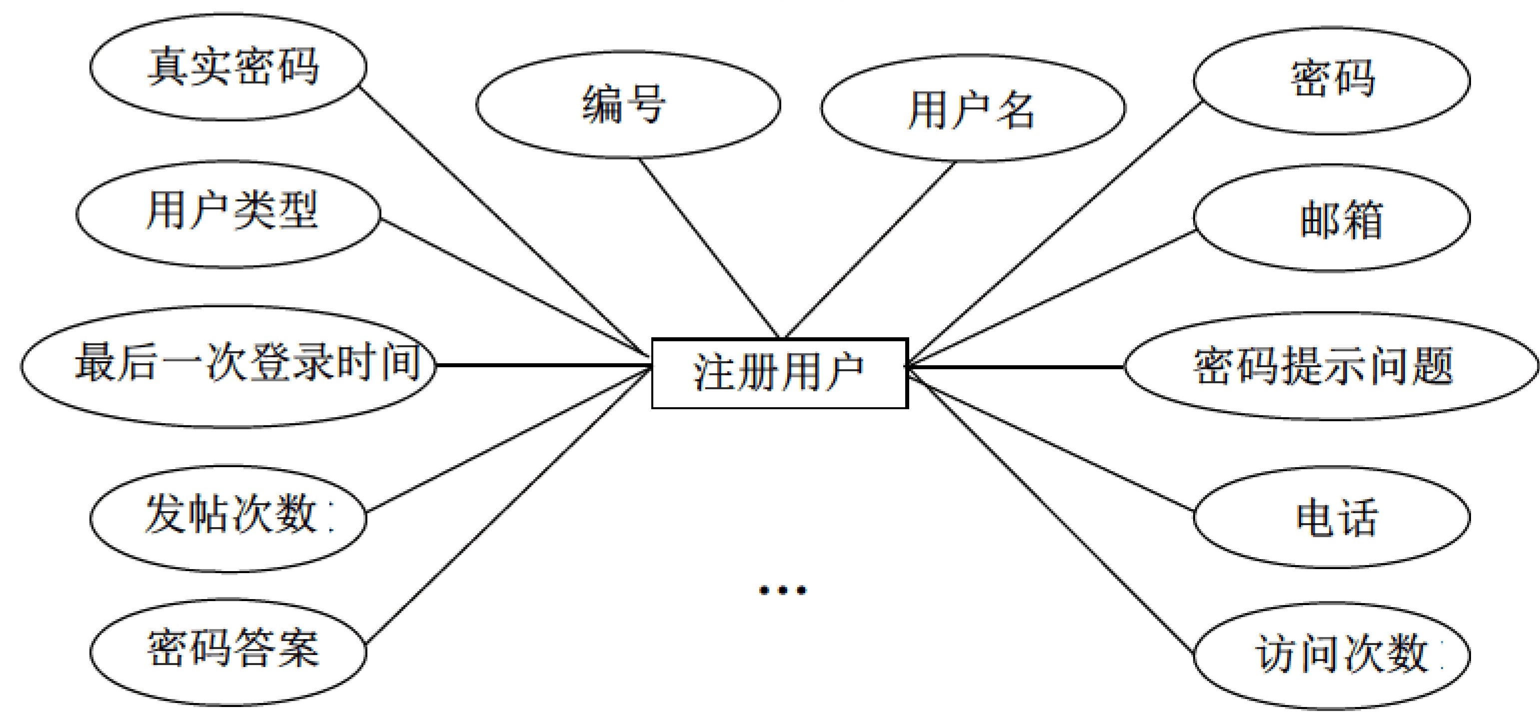


图 22.14　注册用户实体 E-R 图

2．发帖信息实体

发帖信息实体用于存储登录本社区的会员在论坛中发布帖子的相关信息，包括编号、用户名 ID、帖子类别、帖子标题、帖子内容、发帖时间、最后回复时间、表情图、访问次数、是否顶帖、上传图片属性。发帖信息实体的 E-R 图如图 22.15 所示。

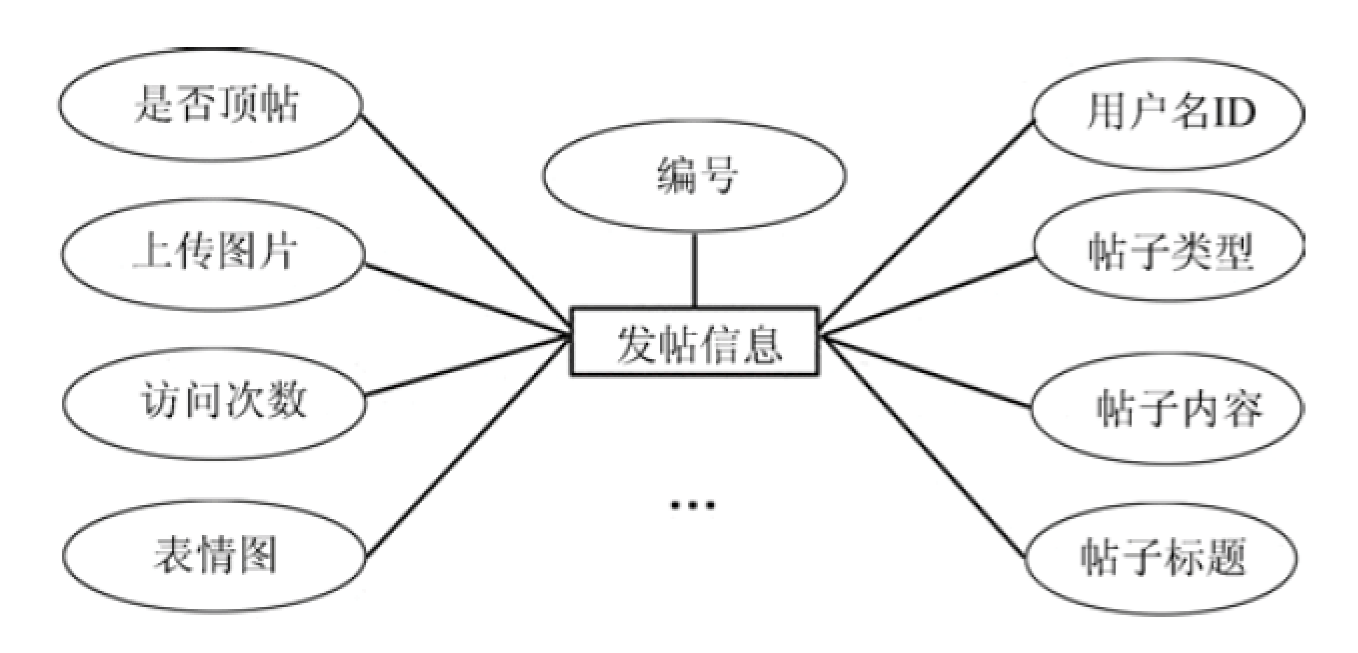


图 22.15　发帖信息实体的 E-R 图

3．订单信息实体

订单信息实体存储用户在线购买时填写的订单信息，包括编号、用户名、性别、家庭地址、邮政编码、QQ 号码、邮箱、手机号码、电话号码、收货方式、邮资、产品金额、订单时间、订单号、选择城市等属性。订单信息实体的 E-R 图如图 22.16 所示。

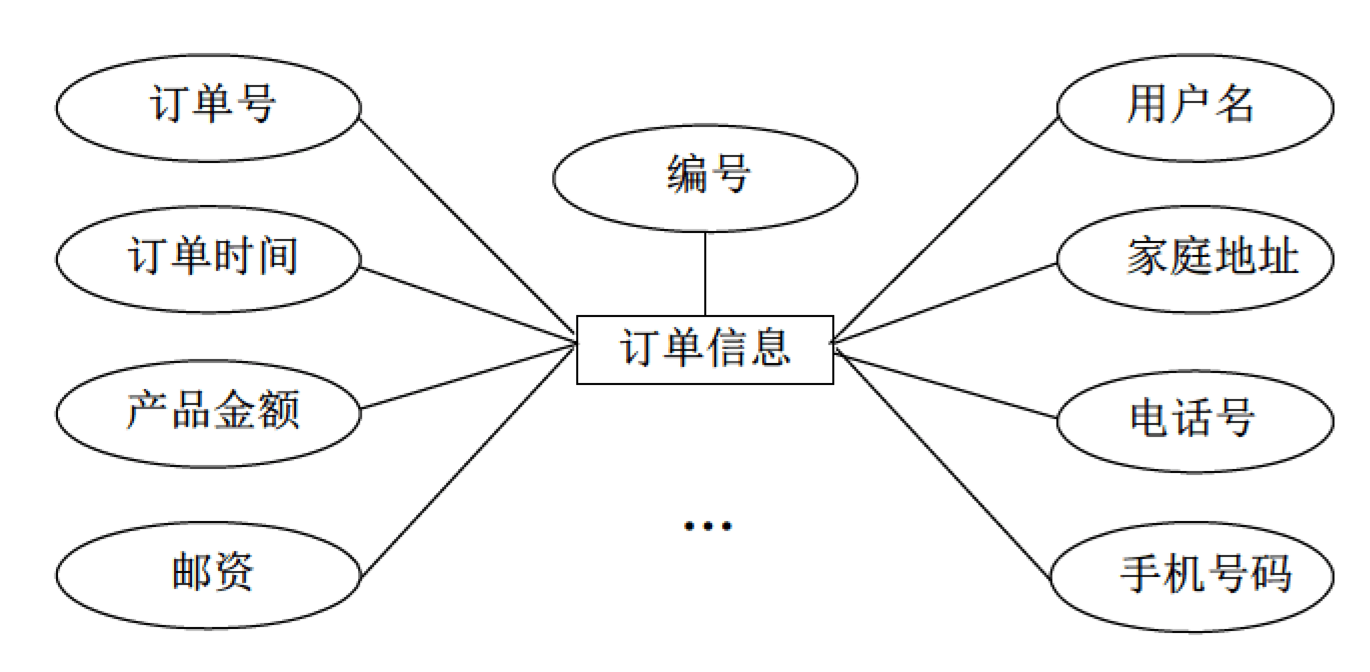


图 22.16　订单信息实体 E-R 图

22.5.3　创建数据库及数据表

在 BCTY365 网上社区系统中应用的是 db_bcty365 数据库，其中涉及 18 个数据表，数据表的名称和功能如图 22.17 所示。

服务器: localhost ▸ 数据库: db_bcty365

表	类型	整理	说明
tb_bb	MyISAM	gb2312_chinese_ci	编程词典版本信息表
tb_bbqb	MyISAM	gb2312_chinese_ci	版本之间区别信息表
tb_bbs	MyISAM	gb2312_chinese_ci	论坛发帖信息表
tb_bccd	MyISAM	gb2312_chinese_ci	编程词典信息表
tb_city	MyISAM	gb2312_chinese_ci	城市信息表
tb_cjwt	MyISAM	gb2312_chinese_ci	常见问题信息表
tb_dd	MyISAM	gb2312_chinese_ci	订单信息表
tb_reply	MyISAM	gb2312_chinese_ci	论坛回帖信息表
tb_sjxz	MyISAM	gb2312_chinese_ci	软件升级下载信息表
tb_soft	MyISAM	gb2312_chinese_ci	软件下载信息表
tb_tell	MyISAM	gb2312_chinese_ci	社区公告信息表
tb_type	MyISAM	gb2312_chinese_ci	社区模块类型信息表
tb_type_big	MyISAM	gb2312_chinese_ci	论坛大类信息表
tb_type_small	MyISAM	gb2312_chinese_ci	论坛小类信息表
tb_user	MyISAM	gb2312_chinese_ci	注册用户信息表
tb_xlh	MyISAM	gb2312_chinese_ci	升级下载序列号信息表
tb_bccdjj	MyISAM	gb2312_chinese_ci	编程词典简介信息表
tb_leaveword	MyISAM	gb2312_chinese_ci	存储客户反馈信息

图 22.17　db_bcty365 数据库中使用的数据表

本案例中创建数据库和数据表使用的是 phpMyAdmin 图形化管理工具。下面将介绍数据库和数据表的创建方法，以及在创建过程中需要注意的一些问题。

1．数据库的创建

打开 phpMyAdmin 图形化管理工具的主页，首先在文本框中输入要创建的数据库的名称（如 db_bcty365），然后在下拉列表框中选择要使用的字符编码格式，这里使用的是 gb2312_chinese_ci，如图 22.18 所示。最后单击“创建”按钮，数据库创建成功。

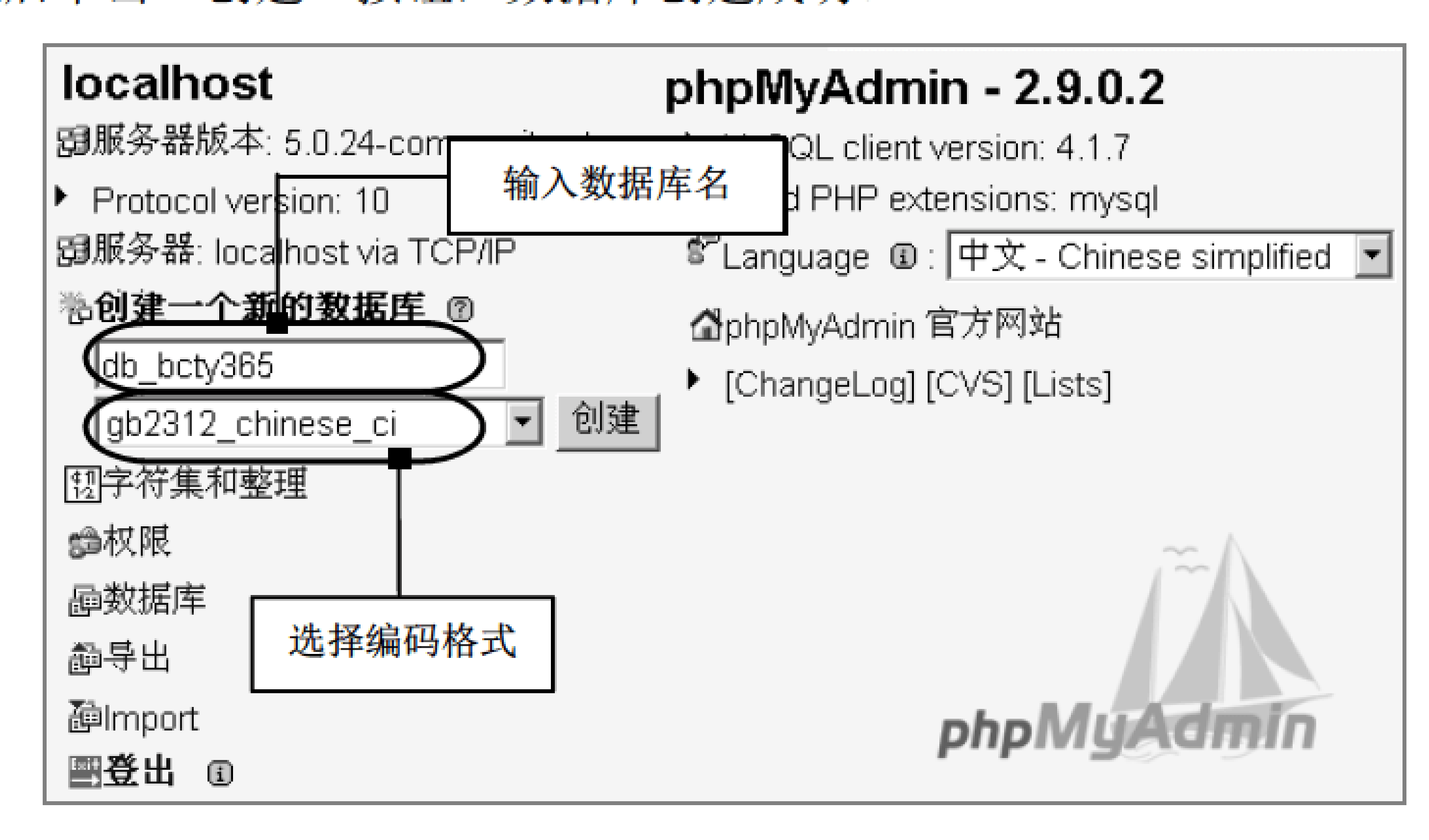


图 22.18　phpMyAdmin 管理界面

技巧

创建数据库的过程中，尽量使用与程序内容贴切的英文字符进行命名，有助于对数据库的理解；如果使用 AppServ 配置 PHP 开发环境，那么在创建数据库时不需要指定编码的格式，默认值为 gb2312_chinese_ci；如果自行配置开发环境，那么就要指定编码格式为 gb2312_chinese_ci，否则创建数据库的编码格式为 latin1_swedish_ci，将导致数据库中数据出现乱码。

2．创建数据表

在成功创建数据库以后，接下来就是创建数据表，这里以 tb_bb 编程词典版本信息表为例，讲解如何创建数据表，以及在创建数据表的过程中都需要注意哪些问题。这里创建一个名称为 tb_bb 的数据表，包括 3 个字段，如图 22.19 所示。

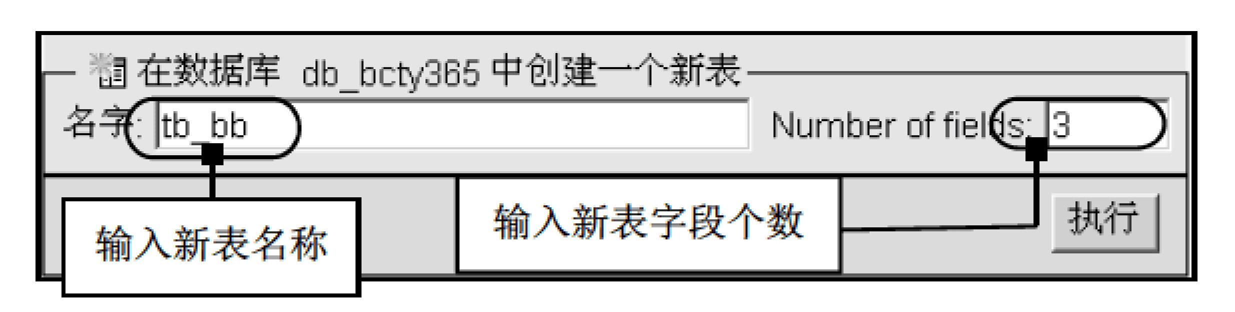


图 22.19　创建 tb_bb 数据表

单击“执行”按钮后，进入到如图 22.20 所示的添加字段信息的页面中，在此处对字段进行详细的设置，包括字段名、数据类型、长度/值、属性、默认值、额外、主键和索引等。

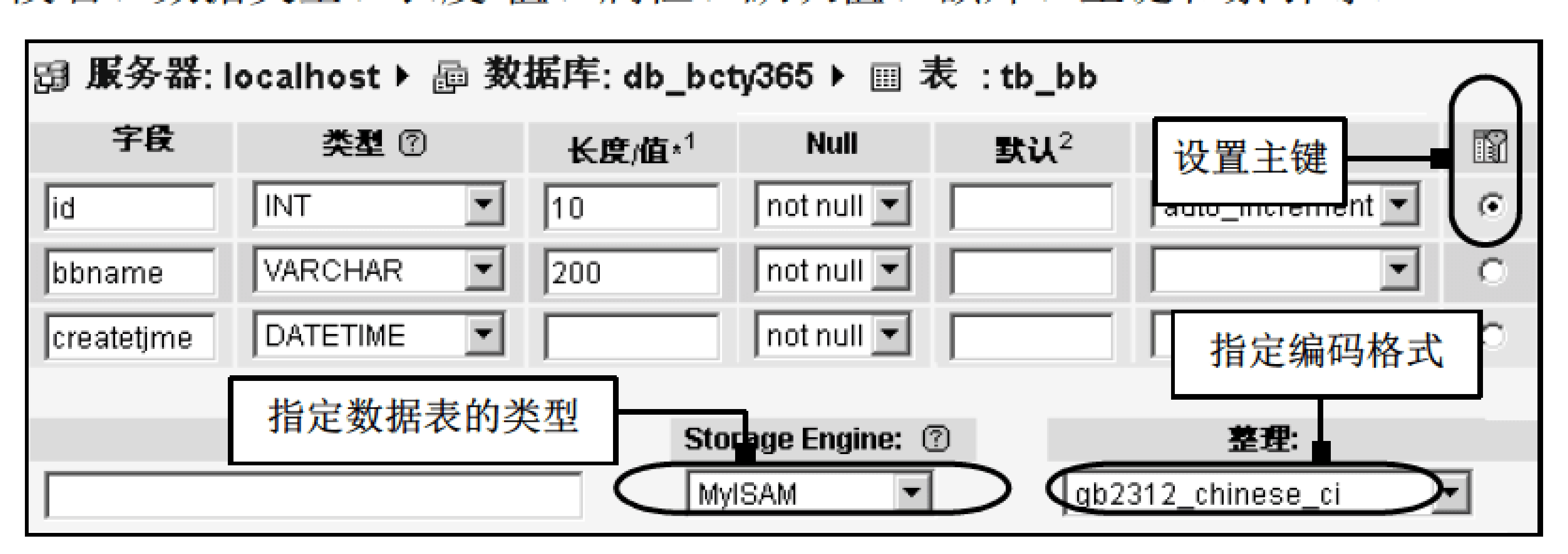


图 22.20　添加数据表中字段信息

技巧

创建数据库中的数据表时，字段名的设计尽量要与数据表的内容相符合，这样有助于程序后期维护和修改工作的进行，能够直观地看出数据表的作用。

如果使用 AppServ 配置 PHP 开发环境，那么在创建数据表时不需要指定数据表类型和编码格式；如果使用自行配置的 PHP 开发环境，那么就要指定数据表的类型为 MyISAM 和字符的编码格式为 gb2312_chinese_ci；否则创建的数据表类型为 InnoDB，而编码格式为 latin1_swedish_ci，将导致该数据表中的数据复制到其他机器上不可用，并且数据表中的数据出现乱码。

在创建数据表的过程中，一定要为数据表指定一个主键，它是数据表的一个唯一的标识。

掌握数据表的创建方法后，就可以自行创建数据表。由于本案例中涉及的数据表有 17 个之多，这

里不能对每个数据表的功能设计进行一一介绍，所以只给出几个重要的数据表的设计效果图供广大读者参考，其他数据表请参见本书附带的光盘。数据表的设计结构如图 22.21~图 22.23 所示。

1）tb_user（注册用户信息表）

注册用户信息表主要用于存储本社区中会员的个人信息。该数据表的结构如图 22.21 所示。

服务器: localhost ▸ 数据库: db_bcty365 ▸ 表 : tb_user

字段	类型	整理	属性	Null	默认	额外	说明
id	int(8)			否		auto_increment	自动编号ID
usernc	varchar(50)	gb2312_chinese_ci		是	NULL		注册用户名
truename	varchar(50)	gb2312_chinese_ci		是	NULL		真实姓名
pwd	varchar(50)	gb2312_chinese_ci		是	NULL		注册密码
email	varchar(50)	gb2312_chinese_ci		是	NULL		有效邮箱地址
sex	varchar(2)	gb2312_chinese_ci		是	NULL		性别
tel	varchar(20)	gb2312_chinese_ci		是	NULL		联系电话
qq	varchar(20)	gb2312_chinese_ci		是	NULL		QQ号码
address	varchar(100)	gb2312_chinese_ci		是	NULL		联系地址
logintimes	int(8)			否			访问次数
regtime	datetime			否			注册时间
lastlogintime	datetime			否			最后一次登录时间
ip	varchar(20)	gb2312_chinese_ci		否			IP地址
yb	varchar(20)	gb2312_chinese_ci		是	NULL		邮政编码
usertype	int(2)			否			用户类型
question	varchar(200)	gb2312_chinese_ci		否			密码提示问题
answer	varchar(200)	gb2312_chinese_ci		否			密码提示答案
truepwd	varchar(200)	gb2312_chinese_ci		否			真实密码
photo	varchar(50)	gb2312_chinese_ci		否			表情图
pubtimes	int(4)			是	0		发帖次数

图 22.21　注册用户信息表结构

2）tb_reply（论坛回帖信息表）

论坛回帖信息表主要用于存储登录会员在本社区中回复帖子的信息。该数据表的结构如图 22.22 所示。

服务器: localhost ▸ 数据库: db_bcty365 ▸ 表 : tb_reply

字段	类型	整理	属性	Null	默认	额外	说明
id	int(8)			否		auto_increment	自动编号ID
userid	int(8)			否	0		注册用户ID
bbsid	int(8)			否	0		发布帖子表ID
title	varchar(200)	gb2312_chinese_ci		是	NULL		回复主题
content	mediumtext	gb2312_chinese_ci		是	NULL		回复内容
createtime	datetime			是	NULL		回复时间
mark	int(2)			是	NULL		回复记录
photo	varchar(80)	gb2312_chinese_ci		是	NULL		回复图片

图 22.22　论坛回帖信息表结构

3）tb_bccd（编程词典信息表）

编程词典信息表主要用于存储本社区的在线订购模块中出售的编程词典的基本信息。该数据表的结构如图 22.23 所示。

服务器: localhost ▸ 数据库: db_bcty365 ▸ 表 : tb_bccd

字段	类型	整理	属性	Null	默认	额外	说明
id	int(8)			否		auto_increment	自动编号ID
bccdname	varchar(200)	gb2312_chinese_ci		否			编程词典名称
owner	varchar(100)	gb2312_chinese_ci		否			开发者
typeid	varchar(50)	gb2312_chinese_ci		否			版本类型
content	mediumtext	gb2312_chinese_ci		否			编程词典简介
samepart	mediumtext	gb2312_chinese_ci		否			软件共同点
addtime	datetime			否			开发时间
imageaddress	varchar(100)	gb2312_chinese_ci		否			界面存储地址
bbid	int(8)			是	NULL		所属版本的ID
price	float			是	NULL		价格

图 22.23　编程词典信息表结构

22.6　公共模块设计

22.6.1　数据库连接文件

在进行程序开发的过程中，有很多地方都涉及数据库的应用，在应用数据库之前首先要与数据库建立连接，因此可以将数据库的连接代码作为一个公共文件进行存储，在需要使用数据库连接文件的地方直接调用该文件即可。无须重复编写相同的代码，既减少了代码的冗余，也便于对数据库连接文件进行修改。在本项目中笔者将数据库的连接代码存储于 conn.php 中。conn.php 文件的代码如下。

代码位置：光盘\TM\sl\22\bcty365\conn\conn.php

```
<?php
	$conn=mysql_connect("localhost","root","root");				//连接数据库服务器
	mysql_select_db("db_bcty365",$conn);					//连接指定的数据库
	mysql_query("set names gb2312");	//指定数据库中字符的编码格式为 gb2312，避免出现中文乱码的问题
?>
```

成功创建 conn.php 文件后，在需要进行数据库的操作程序中，就可以通过 include 或者其他包含语句调用 conn.php 该文件即可，无须再编写连接数据库的程序代码。应用 include 语句包含 conn.php 文件的代码如下。

```
<?php
	include ("conn/conn.php");						//包含数据库文件
 ?>
```



22.6.2　将文本中的字符转换为 HTML 标识符

在输出数据库中数据的过程中，有必要将数据中的一些特殊字符转换为 HTML 标识符，这样可以避免一些麻烦。例如，在输出一个程序的执行代码的过程中，如果不对其进行转换，那么输出的将不是程序的代码，而是程序的执行结果。这里将文本中字符的转换编写到一个自定义函数 unhtml()中，保存到 function.php 文件中，将其作为一个公共模块来使用，当需要使用时直接调用 function.php 文件即可。function.php 文件包含两个自定义函数，即 unhtml()函数和 msubstr()函数，unhtml()函数用于将数据中的特殊字符转换为 HTML 标识符；msubstr()函数用于对字符串进行指定长度的截取。其代码如下。

代码位置：光盘\TM\sl\22\bcty365\function.php

```
<?php
function unhtml($content){                                    //定义自定义函数的名称
    $content=htmlspecialchars($content);                      //转换文本中的特殊字符
        $content=str_replace(chr(13),"<br>",$content);        //替换文本中的换行符
    $content=str_replace(chr(32)," ",$content);          //替换文本中的 
    $content=str_replace("[_[","<",$content];                 //替换文本中的大于号
    $content=str_replace(")_)",">",$content);                 //替换文本中的小于号
    $content=str_replace("|_|"," ",$content);                 //替换文本中的空格
    return trim($content);                                    //删除文本中首尾的空格
}
//定义一个用于截取一段字符串的函数 msubstr()
function msubstr($str,$start,$len){    //$str 指的是字符串，$start 指的是字符串的起始位置，$len 指的是长度
$strlen=$start+$len;                   //用$strlen 存储字符串的总长度（从字符串的起始位置到字符串的总长度）
for($i=0;$i<$strlen,$i++){             //通过 for 循环语句，循环读取字符串
    if(ord(substr($str,$i,1))>0xa0){ //如果字符串中首个字节的 ASCII 序数值大于 0xa0，则表示为汉字
    $tmpstr.=substr($srt,$i,2);        //每次取出两位字符赋给变量$tmpstr，即等于一个汉字
    $i++;                              //变量自加 1
    }else{                             //如果不是汉字，则每次取出一位字符赋给变量$tmpstr
        $tmpstr.=substr($str,$i,1);}
    }
return $tmpstr;                        //输出字符串
}?>
```

22.7　前台首页设计

当今时代，人们都十分重视对事物的第一印象，第一印象基本上就决定了对某个事物的看法和态度，在网络中更是如此，网站给人的第一印象如果不好，那么就会有很多人因此而不去浏览该网站，无论网站的内容是否丰富。可以说网站首页设计的成功与否直接影响着整个网站的发展。

22.7.1 前台首页概述

网站首页是整个网站的脸面，既要突出企业的形象，又要展示出网站强大的功能。如果网站首页设计得非常成功，那无疑是为整个网站的成功增添了一个砝码。BCTY365 网上社区首页的设计以企业的品牌形象为基础，全力打造网站的整体功能，重点推出企业的软件产品，具体内容如下。

（1）网站菜单导航：首页、技术支持、在线订购、社区论坛、软件下载、升级下载、购买须知、联系我们。

（2）用户注册和登录模块：实现用户注册、会员登录、找回密码和修改密码的功能。

（3）网站公告：主要用于发布社区中的一些新消息和重大事件。

（4）编程词典模块：推广企业的软件产品。

（5）软件下载模块：展示企业提供的适用版和免费的软件产品。

（6）常见问题模块：列举出编程中常见问题的解决方案。

（7）社区论坛模块：浏览社区论坛中的部分帖子。

（8）升级下载模块：提供一些软件的升级版本下载。

上述内容就是 BCTY365 网上社区首页中体现出的内容，为了更加直观地了解网上社区首页的设计，这里先预览一下社区首页，该首页在本书光盘中的路径为\TM\02\bcty365\index.php，如图 22.24 所示。



图 22.24　BCTY365 网上社区首页

BCTY365 网上社区首页的设计看上去很复杂，由很多个版块组成，但实现的过程非常简单。总体架构使用一个两行三列表格和一个三行三列的表格，将其分隔成不同的版块，然后使用脚本语句从数据库中读取数据，最后将数据循环输出到页面中，其中网站的头尾文件使用 include 包含语句调用。首页的框架结构如图 22.25 所示。



图 22.25　网站首页的框架结构

22.7.2　前台首页技术分析

作为网站首页，不一定要具有什么特殊的技术或者功能，应该是以简洁、鲜明，突出企业形象，展示网站的功能为主。即使使用的是一个静态页面，只要能够将内容表达全面、完整，那么这个首页设计也是非常成功的。

在本案例首页的设计中，应用到一个文字循环滚动的技术，通过该技术来输出社区中发布的公告信息。该技术是通过 JavaScript 脚本和 div 标签来共同实现的，其实现的原理是：首先创建一个 div 标签，然后在 div 标签中输出公告信息，最后通过 JavaScript 来对 div 标签进行操作，实现 div 标签中内容的滚动输出。该技术的实现在 index.php 页中完成，其中使用的 JavaScript 脚本的代码如下。

代码位置：光盘\TM\sl\22\bcty365\index.php

```
<script language="JavaScript">
    marqueesHeight=222;                                    //定义输出标签的高度
    stopscroll=false;                                      //定义 stopscroll 的默认值为 false
    with(marquees){                                        //编辑 marquees 标签的属性
        style.width=0;                                     //定义初始宽为 0
        style.height=marqueesHeight;                       //定义 marquees 标签的高为 222
        style.overflowX="visible";                         //定义值为显示
        style.overflowY="hidden";                          //定义值为隐藏
        noWrap=true;
        onmouseover=new Function("stopscroll=true");       //当鼠标经过时执行 stopscroll=true
```

```
        onmouseout=new Function("stopscroll=false");           //当鼠标离开时执行 stopscroll=false
    }
    //创建一个新的 div"templayer"与 div"marquees"进行连接，实现不间断地循环输出内容
    document.write('<div id="templayer" style="position:absolute;z-index:1;visibility:hidden"></div>');
    preTop=0; currentTop=0;
    function init(){
        templayer.innerHTML="";                                //设置 templayer 的初始值为空
        while(templayer.offsetHeight<marqueesHeight){   //判断当 templayer 的高度小于 marquees 的高度时
            templayer.innerHTML+=marquees.innerHTML;         //将 templayer 的值赋给 marquees
        }
        marquees.innerHTML=templayer.innerHTML+templayer.innerHTML;       //将 templayer 的值累加
        setInterval("scrollup()",50);                          //间隔 50 毫秒执行一次 scrollup()函数
    }
    function scrollup(){                                       //实现滚动输出
        if(stopscroll==true) return;                           //判断如果 stopscroll==true，不执行循环
        preTop=marquees.scrollTop;
        marquees.scrollTop+=1;
        if(preTop==marquees.scrollTop){
            marquees.scrollTop=templayer.offsetHeight-marqueesHeight;
            marquees.scrollTop+=1;
        }
    }
</script>
```

在 div 标签中，主要是输出数据库中存储的公告信息，并且对输出的信息进行截取和替换，规范输出的内容。div 标签中的程序代码如下。

```
<div id=marquees >  <!-->创建一个 div 标签<!-->
<table width="200" height="25" border="0" align="center"   cellpadding="0" cellspacing="0">
    <?php                                                       //从数据库中读取公告数据
        $sql=mysql_query("select id,title,createtime from tb_tell order by createtime desc limit 0,10",$conn);
        $info=mysql_fetch_array($sql);
        if($info==false){                                      //判断当$info==false 时执行下面的内容
    ?>
    <tr>
    <td height="25"><div align="center"><a href="#" class="a4">本站暂无公告发布! </a></div></td>
   </tr>
    <?php     }else{
        $i=1;                                                  //定义变量$i=1
        do{                                                    //执行 do…while 循环语句
    ?>
   <tr>
    <td height="25" style="padding:4">
            <a href="tellinfo.php?id=<?php echo $info[id];?>" class="a1">
   <?php
        if($i==1){                                             //判断当$i==1 时,将输出的内容设置为红色
            echo "<font color=red>";
        }
        echo $i.". ";
```

```
            echo unhtml(msubstr($info[title],0,50));              //应用自定义函数对输出的内容进行控制
            if(strlen($info[title])>50){                          //当输出内容的长度超过 50 个字符时用"..."代替
                echo " ...";
            }
            echo "(".str_replace("-","/",$info[createtime]).")";  //将输出的公告时间中的"-"用"/"替代
            if($i==1){  echo "</font>"; }
        ?>          </a>
            </td>
        </tr>
        <?php
            $i++;
            }while($info=mysql_fetch_array($sql));                //do…while 循环语句结束
            }
        ?>
</table>
    </div>
```

在首页中使用滚动条是一个比较不错的方法，通过其可以增加网页的动态效果，增加网页的观赏性，而且不会影响到网页的浏览速度。

22.7.3　前台首页的实现过程

开发网站首页主要就是连接数据库，从数据库中读取数据，最后应用循环语句将数据库中数据输出到前台页面。由于使用的代码较多，而且多数都是重复使用，所以这里只给出首页中公告发布模块的代码。

公告发布模块主要实现从数据库中读取公告数据，将数据在首页中滚动输出，并且对公告信息的长度进行控制，保证内容的整齐、规范。具体详细代码可以参考本书光盘中 TM\02\bcty365\index.php 文件，index.php 文件的部分代码如下。

代码位置：光盘\TM\sl\22\bcty365\index.php

```
<?php include_once("top.php");                       //获取头部文件  ?>
…//省略部分代码
<?php
    //读取数据库中公告表中的数据
    $sql=mysql_query("select id,title,createtime from tb_tell order by createtime desc limit 0,10",$conn);
    $info=mysql_fetch_array($sql);                   //执行读取数据表中数据的语句
    if($info==false){                                //如果返回值为空则执行下面的语句
?>
<tr>
    <td height="25"><div align="center"><a href="#" class="a4">本站暂无公告发布! </a></div></td>
</tr>
<?php
    }else{                                           //如果返回值不为空则执行下面的 do…while 循环语句
        $i=1;
        do{
```

```
?>
<tr>
    <td height="25" style="padding:4"><a href="tellinfo.php?id=<?php echo $info[id];?>" class="a1">
<?php
    if($i==1){                                          //判断当变量$i=1 时，输出的内容以红色字体显示
        echo "<font color=red>";
    }
    echo $i.". ";
    echo unhtml(msubstr($info[title],0,50));
    if(strlen($info[title])>50){                        //如果标题长度大于 50 个字符，则以省略号代替
        echo " ...";
    }
    echo "(".str_replace("-","/",$info[createtime]).")";    //输出公告发布的时间，并且将其中的"/"使用"-"替换
    if($i==1){
        echo "</font>";
    }
?>
        </a>
    </td>
</tr>
<?php
$i++;                                                   //变量自加 1
}while($info=mysql_fetch_array($sql));                  //do…while 循环语句结束
}
?>
…                                                       //省略了部分代码
<?php
include_once("bottom.php");                             //调用网站的尾文件
?>
```

22.8　注册模块设计

22.8.1　注册模块概述

BCTY365 网上社区系统为了更好地与广大网民朋友进行交流和沟通，创建了一个会员注册模块。通过会员注册模块，可以有效地对用户信息进行采集，并将合法的用户信息保存到指定的数据表中，实现与用户的长期沟通和交流。既然设置了会员注册模块，那么在系统中就要为会员提供一些特殊的权限。在本系统中注册会员可以拥有如下权限：在本社区的论坛中发布和回复帖子；在技术支持模块中发表留言；在升级下载模块中下载软件升级包等；而且可以进行修改密码和找回密码。用户注册模块的运行结果如图 22.26 所示。

用户录入 版块

您已经成功地申请了本站的用户名： PHP程序员 如果注册为本站的用户，您还应该填写以下信息！

注意：以下注册信息均为必添内容！

真实姓名： 潘**

性别： 男

邮箱地址： pkh@sina.com （为了便于工作人员与您联系，请填写正确的E-mail地址！）

联系电话： 1360433****

邮编： 130025 （邮编只能由数字组成，并且为6位！）

QQ号码： 18381769 （QQ号只能由数字组成！）

家庭住址： 长春市

头像选择： 4.gif

密码保护问题： 我最喜欢的休闲运动是什么？

您的答案： 打羽毛球 （为了能够找回丢失的密码，请记住该答案！）

效验码： 5861 5861

提交注册信息　重新填写注册信息

图 22.26　用户注册模块的运行结果

22.8.2　注册模块技术分析

在会员注册模块中，必不可少的就是要对用户输入的信息进行判断，首先判断用户填写的注册信息中哪些是必须填写的，哪些可以不填写，然后进一步判断输入的信息是否合理合法，例如，判断输入的邮编的格式是否正确，判断输入邮箱的格式是否正确等。对表单中提交数据进行判断最常用的办法就是使用 JavaScript 脚本，也可以使用正则表达式。下面讲解在本模块中是如何通过 JavaScript 实现表单提交数据验证。

操作原理是：在 form 表单中调用 onsubmit 事件，通过该事件调用指定的 JavaScript 脚本，执行 chkinput()自定义函数，实现对表单中提交数据的验证。在 JavaScript 脚本中，实现对表单中提交数据进行判断，判断输入的内容是否为空，判断内容的格式是否正确，如果正确则继续执行；否则将弹出提示对话框，并将鼠标的焦点指定到出错的位置。具体的 JavaScript 脚本代码如下。

```
<script language="JavaScript" type="text/javascript">
    function chkinput(form){                                  //定义一个函数
        if(form.tel.value==""){                               //判断 tel 文本框中的值是否为空
            alert("请填写联系电话!");                          //如果为空则输出"请填写联系电话!"
            form.tel.select();                                //返回到 tel 文本框
            return(false);
        }
        if(form.email.value==""){                             //判断 email 文本框的值是否为空
        alert("请输入 E-mail 地址!");                          //如果为空则输出"请输入 E-mail 地址!"
            form.email.select();                              //返回到 email 文本框
            return(false);
        }
```

```
            var i=form.email.value.indexOf("@");
            var j=form.email.value.indexOf(".");
             //进一步判断邮箱的格式是否正确,是否包含"@"和"."
            if((i<0)||(i-j>0)||(j<0)){
                alert("请输入正确的 E-mail 地址!");
                 form.email.select();                                      //返回到 email 文本框
                 return(false);
            }
        return(true);                                                      //提交表单
    }
</script>
```

上述代码中，只是列举了 JavaScript 中的部分内容，并且在对电话号码进行判断时，只是判断其是否为空，没有进一步判断电话号码的格式是否正确。如果想要更加准确地判断电话号码的格式是否正确，可以采用下面的方法：通过正则表达式的 preg_match()函数，在表单提交处理页中对电话号码进行判断。

preg_match()函数的语法格式如下。

```
int preg_match ( string pattern, string subject [, array matches [, int flags]] )
```

preg_match()函数的参数说明如表 22.1 所示。

表 22.1　preg_match()函数的参数说明

参　　数	说　　明
pattern	必要参数。需要匹配的正则表达式
subject	必要参数。输入的字符串
matches	可选参数。输出的搜索结果的数组，如$out[0]将包含与整个模式匹配的结果，$out[1]将包含与第一个捕获的括号中的子模式所匹配的结果，以此类推
flags	可选参数。标记：PREG_OFFSET_CAPTURE，对每个出现的匹配结果也同时返回其附属的字符串偏移量，本标记自 PHP 4.3.0 起可用

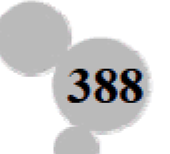
通过 preg_match()函数判断电话号码的格式是否正确的方法如下：首先定义一个用于判断电话号码格式的正则表达式。代码如下。

```
/^(\d{3}-)(\d{8})$|^(\d{4}-)(\d{7})$|^(\d{4}-)(\d{8})$/
```

正则表达式的功能分析如下：使用“^”和“$”对字符串进行边界的限制，对区号从字符串的开始进行匹配，对其他号码从字符串的末尾开始进行匹配；将括号“()”中的内容作为一个原子使用；使用“\d”来匹配一个数字，区号为 3 或 4 个数字，其他数字为 7 或 8 个；使用“{}”来对前字符进行重复匹配；使用“|”对匹配的模式进行选择，分成 3 个模式。

然后将该正则表达式应用到 preg_match()函数中，对表单提交的电话号码进行判断，如果正确则继续执行；否则弹出提示信息，并返回到表单提交页，代码如下。

```
  <?php
     $tel="0431-84978981";                                                 //定义一个电话号码的变量
     if(preg_match("/^(\d{3}-)(\d{8})$|^(\d{4}-)(\d{7})$|^(\d{4}-)(\d{8})$|^(\d{11})$/",$tel,$counts)){
            echo "您输入的电话号码格式正确!";                               //输出字符串
```



```
    }else{
        echo "<script>alert('您输入的电话号码的格式不正确!!');history.back()</script>";
    }
    ?>
```

22.8.3　注册模块的实现过程

注册模块的实现过程非常简单，首先阅读注册服务条款，然后填写用户注册的用户名和密码，提交后由系统判断输入的用户名是否被占用，如果未被占用则可以继续注册，填写详细的注册信息，将数据提交到表单处理页进行处理，最后将用户注册的信息保存到指定的数据表中。用户注册模块的实现过程主要由 3 个文件完成：register.php 用于输出注册服务条款，以及填写注册的用户名和密码，并且判断注册的用户名和密码是否被占用；getuserinfo.php 文件用于填写详细的注册信息，并且在表单中应用数字验证码技术；savereginfo.php 文件用于对表单中提交的数据进行处理，将数据保存到指定的数据表中。

在 savereginfo.php 文件中，首先连接数据库，然后获取到表单中提交的数据，并且判断提交的用户名是否被占用，最后将提交的数据进行处理，并将数据保存到指定的数据表中。程序代码如下。

代码位置：光盘\TM\sl\22\bcty365\saverreginfo.php

```
<?php session_start();                                          //初始化 session 变量
include_once("conn/conn.php");                                  //连接数据库
$usernc=trim($_POST[usernc]);                                   //获取注册的用户名
//判断指定的用户名是否存在
$sql=mysql_query("select usernc from tb_user where usernc='".$usernc."'",$conn);
$info=mysql_fetch_array($sql);                                  //按指定条件检索数据信息
if($info!=false){                                               //如果查询结果不为空，则执行以下操作
    echo "<script language='javascript'>alert('对不起，该昵称已被其他用户使用!');history.back();</script>";
    exit;
}
$xym=trim($_POST[xym]);                                         //去除变量两边的空格
$num=$_POST[num];                                               //接收变量值
if(strval($xym)!=strval($num)){
    echo "<script>alert('验证码输入错误!');window.location.href='register.php';</script>";
    exit;}
//对表单提交的数据进行处理
$truepwd=trim($_POST[pwd1]);                                    //获取真实密码
$pwd=md5($truepwd);                                             //获取加密密码
$truename=trim($_POST[truename]);                               //获取真实姓名
$email=trim($_POST[email]);                                     //获取邮箱地址
$sex=$_POST[sex];                                               //获取性别
$tel=trim($_POST[tel]);                                         //获取电话
$yb=trim($_POST[yb]);                                           //获取邮政编码
$qq=trim($_POST[qq]);                                           //获取 QQ
$address=trim($_POST[address]);                                 //获取地址
$question=trim($_POST[question]);                               //获取提示问题
$answer=trim($_POST[answer]);                                   //获取问题答案
```

```
$ip=getenv("REMOTE_ADDR");                                          //获取客户端的 IP
$logintimes=1;                                                      //指定访问次数
$regtime=date("Y-m-j H:i:s");                                       //获取时间
$lastlogintime=$regtime;
$usertype=0;                                                        //指定用户类型，默认为 0
$photo=$_POST["photo"];                                             //获取头像
//将表单中提交的数据存储到数据表中
if(mysql_query("insert into
tb_user(usernc,truename,pwd,email,sex,tel,qq,address,logintimes,regtime,lastlogintime,ip,usertype,yb,question
,answer,truepwd,photo)
values('$usernc','$truename','$pwd','$email','$sex','$tel','$qq','$address','$logintimes','$regtime','$lastlogintime','
$ip','$usertype','$yb','$question','$answer','$truepwd','$photo')",$conn)){
    session_register("unc");
    $_SESSION["unc"]=$usernc;
    echo "<script>alert('注册成功!');window.location.href='index.php';</script>";
}else{                                                              //如果添加操作失败，则给出提示
    echo "<script language='javascript'>alert('对不起,注册失败!');history.back();</script>";
    exit;                                                           //退出
}
?>
```

22.9　技术支持模块设计

技术支持模块主要是从浏览者的角度进行设计，存储大量技术问题的解决方案数据，为浏览者查阅提供方便，而且设计一个企业与客户沟通的平台，能够随时了解客户或者会员的意见和需求。

22.9.1　技术支持模块概述

技术支持模块主要由 3 个子模块组成，包括常见问题、客户反馈和联系方式。常见问题模块主要用于展示编程中一些常见问题的解决方案或者方法，为浏览者解决编程中的疑难问题提供方便；客户反馈模块主要用于收集和获取来自客户的需求和意见；联系方式模块主要用于展示企业的形象和具体的联系方式。

22.9.2　技术支持模块技术分析

技术支持模块中在对常见问题解决方案的数据进行输出时，使用了分页处理技术，该技术的设计思路是：从数据库中读取数据，获取数据总量，在每页中显示 20 条数据，根据数据总量和每页显示的条数对数据进行分页处理，计算出有多少页和当前显示的页码，实现首页、上一页、下一页和尾页之间的页面跳转。具体的设计思路可以参考 cjwt.php 文件中的代码注释和代码贴士。cjwt.php 文件的程序代码如下。

代码位置：光盘\TM\sl\22\bcty365\cjwt.php

```
<?php
	$sql=mysql_query("select count(*) as total from tb_cjwt",$conn);	//读取数据库中数据
	$info=mysql_fetch_array($sql);					//返回数据
	$total=$info[total];
//判断字段 total 是否为空，为空则执行下面的内容
	if($total==0){
?>
	<tr>
	<td height="22" colspan="2"><div align="center">对不起，暂无常见问题!</div></td>
	</tr>
<?php
	}else{								//如果不为空，则执行下面的内容
		if(!isset($_GET["page"]) || !is_numeric($_GET["page"])){	//判断$_GET 获取的 page 的值是否存在
			$page=1;					//如果不存在，则设置变量的值为 1
		}else{
			$page=intval($_GET["page"]);			//如果存在，则获取变量$_GET 的值
		}
		//设置变量$pagesize,每页显示的数据数据量为 20
		$pagesize=20;
		if($total%$pagesize==0){				/如果变量的值为 0
			$pagecount=intval($total/$pagesize);		//获取变量的整数值
		}else{
			$pagecount=ceil($total/$pagesize);		//如果不为 0 则获取实际的整数值
		}
		//读取数据库中数据，按照时间进行降幂排列
		$sql=mysql_query("select * from tb_cjwt order by createtime desc limit ".($page-1)*$pagesize.",
$pagesize  ",$conn);
		while($info=mysql_fetch_array($sql)){
?>
…									//省略部分代码
<table width="600" height="25" border="0" align="center" cellpadding="0" cellspacing="0">
	<tr>
	<td width="479"><div align="left">  共有常见问题 <?php echo $total;?> 条 
每页显示 <?php echo $pagesize;?> 条 第 <?php echo $page;?> 页/共
 <?php echo $pagecount;?> 页</div></td>
	  <td width="269"><div align="right">
		<a href="<?php echo $_SERVER["PHP_SELF"]?>?page=1" class="a1">首页</a> 
		<a href="<?php echo $_SERVER["PHP_SELF"]?>?page=<?php
			if($page>1)					//判断如果页码大于 1
				echo $page-1;				//输出前一页
			else
				echo 1;
				?>" class="a1">上一页</a> 
		<a href="<?php echo $_SERVER["PHP_SELF"]?>?page=<?php
		if($page<$pagecount)					//如果页码小于总页数
			echo $page+1;
		else
			echo $pagecount;				//输出下一页
				?>" class="a1">下一页</a> 
		<a  href="<?php  echo  $_SERVER["PHP_SELF"]?>?page=<?php  echo  $pagecount;?>"
```

```
class="a1">尾页</a>  </div>
            </td>
        </tr>
    </table>
```

22.9.3 常见问题的实现过程

常见问题子模块实现的主要功能是展示出数据库中存储的有关编程中遇到的常见问题及解决方案。其运行结果如图 22.27 所示。



图 22.27 常见问题模块的运行结果

该模块由两个文件组成，一个是 cjwt.php 文件，用于存储创建问题数据，详细内容可以参考 22.9.2 节；另一个是 lookcjwt.php 文件，用于输出 cjwt.php 文件中对应问题的详细介绍和解决方案。其代码如下。

代码位置：光盘\TM\sl\22\bcty365\lookcjwt.php

```
<?php
include_once("conn/conn.php");                                    //与数据库建立连接
include_once("function.php");                                     //调用自定义函数
//读取 tb_cjwt 表中的数据,条件为 id="$_GET[id]"
$sql=mysql_query("select * from tb_cjwt where id='".$_GET["id"]."'",$conn);
$info=mysql_fetch_array($sql);
?>
<table width="635" height="100" border="0" align="center" cellpadding="0" cellspacing="0">
      <tr>
      <td width="94" height="30"><div align="center"><strong>问  题：</strong></div></td>
          <td width="541"><?php echo unhtml($info["question"]);       //输出问题的详细内容 ?></td>
      </tr>
      <tr>
            <td height="70"><div align="center"><strong>解  答：</strong></div></td>
          <td height="70"> <?php echo unhtml($info["answer"]);    //输出问题的解决方案 ?></td>
      </tr>
</table>
```

22.9.4　客户反馈的实现过程

客户反馈子模块为客户提供一个反馈意见和提出要求的平台，并且将提交的信息存储到数据库中。其运行结果如图 22.28 所示。

图 22.28　客户反馈模块的运行结果

该功能只对本网站中的会员开通，即只有以会员身份登录的用户才具有反馈信息的权限，其中在对提交表单的细节处理上使用 JavaScript 脚本来验证表单中的值是否为空，而且还使用数字验证码技术。对表单中提交的数据进行处理是在 saveleaveword.php 文件中完成，该文件主要用于对表单中提交的数据进行处理，将数据存储到数据库中，其代码如下。

代码位置：光盘\TM\sl\22\bcty365\saveleaveword.php

```
<?php session_start();                                                    //初始化一个 session 变量
$xym=$_POST[xym];                                                         //获取$_POST 提交的值
if($xym!=$_SESSION["autonum"]){                                           //判断验证码是否正确
    echo "<script>alert('效验码输入错误!');history.back();</script>";
    exit;
}
$title=$_POST["title"];                                                   //获取反馈信息的标题
$content=$_POST["content"];                                               //获取反馈信息的内容
$type=$_POST["type"];                                                     //获取反馈信息的类型
include_once("conn/conn.php");                                            //与数据库建立连接
$sql=mysql_query("select id from tb_user where usernc='".$_SESSION["unc"]."'",$conn); //读取数据库中数据
$info=mysql_fetch_array($sql);                                            //获取结果集中的数组
$userid=$info["id"];
//向数据库中添加数据
if(mysql_query("insert into tb_leaveword(userid,type,title,content,createtime)
values('$userid','$type','$title','$content','".date("Y-m-j H:i:s")."')",$conn)){
    echo "<script>alert('留言发表成功!');history.back();</script>";       //添加操作成功，给出提示信息
}else{
    echo "<script>alert('留言发表失败!');history.back();</script>";       //添加操作失败，给出提示信息
}
?>
```

22.10　在线订购模块设计

在线订购模块的功能是实现在线购买企业推出的软件产品，其操作的流程主要通过购物车和订单管理来实现。

22.10.1　在线订购模块概述

在线订购的功能对所有访问网站的人开放，没有任何的权限限制。其操作流程如图 22.29 所示。

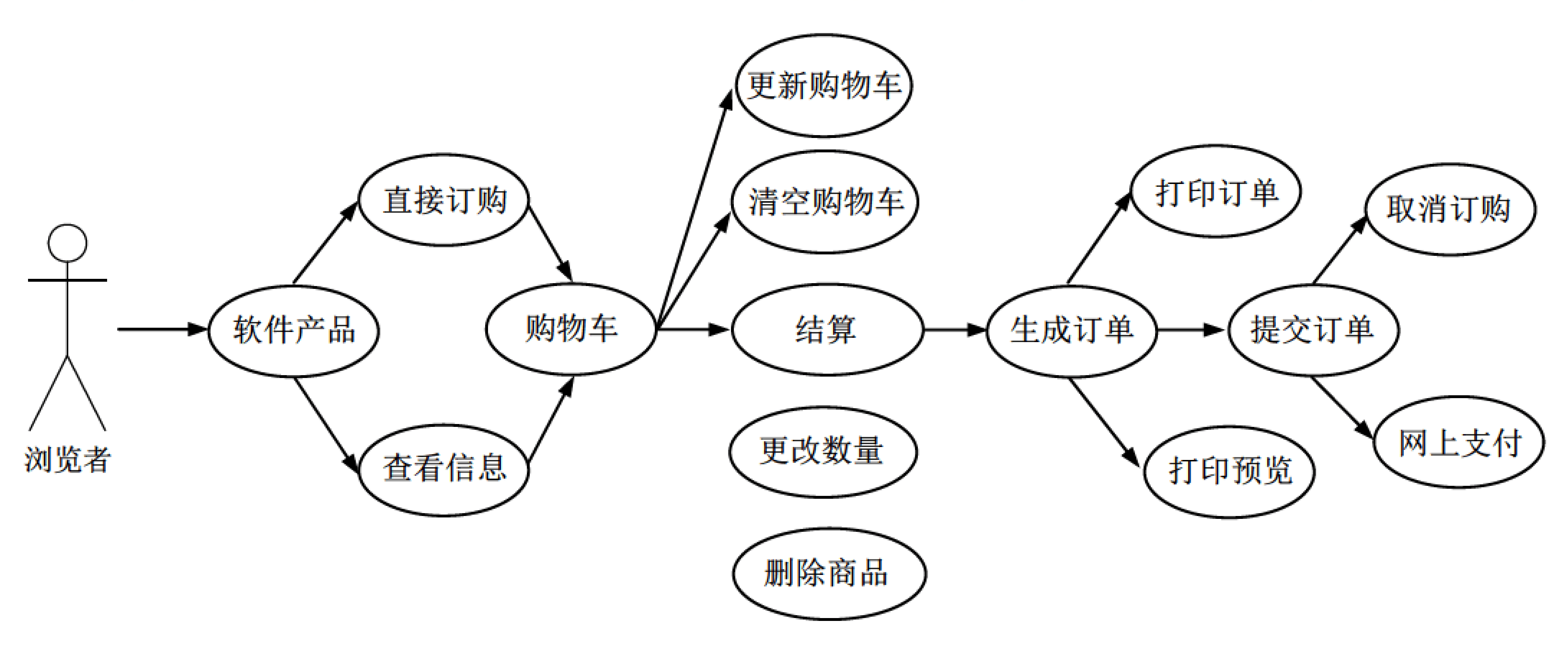


图 22.29　在线订购模块的操作流程

22.10.2　在线订购模块技术分析

在线订购管理模块中，不可缺少的一项内容就是对订单进行打印。下面就来讲解一下订单打印功能的实现方法。在线订购管理模块中运用的是 WebBrowser 打印方法。WebBrowser 是 IE 内置的浏览器控件，无须用户下载。其优点是客户端独立完成打印目标文档的生成，减轻服务器负荷；缺点是源文档的分析操作复杂，并且要对源文档中要打印的内容进行约束。

下面介绍 WebBrowser 控件的具体参数，如表 22.2 所示。

表 22.2　WebBrowser 控件的具体参数说明

参 数 名 称	说　　明
document.all.WebBrowser.Execwb(7,1)：	表示打印预览
document.all.WebBrowser.Execwb(6,1)：	表示打印
document.all.WebBrowser.Execwb(6,6)：	表示直接打印

续表

参 数 名 称	说 明
document.all.WebBrowser.Execwb(8,1)：	表示页面设置
document.all.WebBrowser.Execwb(1,1)：	打开页面
document.all.WebBrowser.Execwb(2,1)：	关闭所有打开的 IE 窗口
document.all.WebBrowser.Execwb(4,1)：	保存网页
document.all.WebBrowser.Execwb(10,1)：	查看页面属性
document.all.WebBrowser.Execwb(17,1)：	全选
document.all.WebBrowser.Execwb(22,1)：	刷新
document.all.WebBrowser.Execwb(45,1)：	关闭窗体无提示

该技术的实现原理是：首先通过 onClick 事件调用一个 JavaScript 脚本，然后执行 openprintwindow()函数，将指定的变量值传递到订单打印页中（printwindow.php），最后在订单打印页中实现打印及打印预览的功能。调用 JavaScript 脚本和执行 openprintwindow()函数在 shopping_dd.php 页中完成。其关键代码如下。

代码位置：光盘\TM\sl\22\bcty365\shopping_dd.php

```
<script language="javascript">
      function openprintwindow(x,y,z){                                        //定义一个函数，获取传递的参数
      //通过 window 对象中的 open 方法打开一个新窗口，并设置其属性
      window.open("printwindow.php?ddno="+x+"&pv="+z,"newframe","top=200,left=200,width=635,
height="+(230+20*y)+",menubar=no,location=no,toolbar=no,scrollbars=no,status=no");
        }
      </script>
<!-- ---------通过 onclick 事件调用 JavaScript 脚本，传递参数值。当参数 z 的值为"p"时执行打印功能----------- -->
<td width="75"> <img src="images/bg_14(11).jpg" width="69" height="20"
        onclick="javascript:openprintwindow('<?php echo base64_decode($_GET["ddno"]);?>','<?php echo
$gnum;?>','p')" style="cursor:hand"/></td>
<!-- ------通过 onclick 事件调用 JavaScript 脚本，传递参数值。当参数 z 的值为"v"时执行打印预览功能--------- -->
<td width="90"><img src="images/bg_14(12).jpg" width="69" height="20"
onclick="javascript:openprintwindow('<?php echo base64_decode($_GET["ddno"]);?>','<?php echo
$gnum;?>','v')"
style="cursor:hand"/></td>
```

订单的打印和打印预览的功能在 printwindow.php 页中完成，首先编写一个实现打印预览功能的 JavaScript 脚本，然后建立 HTML 的 object 标签，调用 WebBrowser 控件，最后获取变量传递的值，当变量的值为 p 时执行打印功能；当变量的值为 v 时执行打印预览的功能。printwindow.php 页的关键代码如下。

代码位置：光盘\TM\sl\22\bcty365\printwindow.php

```
<script>
      function printview(){                                   //定义一个函数
            document.all.WebBrowser1.ExecWB(7,1) ;             //执行 WebBrowser 控件，实现打印预览
            window.close();                                   //关闭窗口
      }
</script>
```

```
<!-- ----------------------------------建立 HTML 的 object 标签，调用 WebBrowser 控件-------------------------- -->
<object ID='WebBrowser1' WIDTH=0 HEIGHT=0
CLASSID='CLSID:8856F961-340A-11D0-A96B-00C04FD705A2'></object>
<!--- ------------------------------------在 body 中调用 onload 事件，执行打印操作------------------------------------ -->
    <body topmargin="0" leftmargin="0" bottommargin="0" onLoad="
<?php
    if($_GET["pv"]=="p"){      //判断当变量值为"p"时，执行打印操作
?>
    window.print();
<?php
    }elseif($_GET["pv"]=="v"){//判断当变量值为"v"时，执行打印预览操作
?>
    printview()
<?php }    ?>
">
```

订单打印操作的运行结果如图 22.30 所示。



图 22.30 订单打印操作的运行结果

22.10.3 购物车的实现过程

购物车的功能是临时储存用户选购的商品，用户可以对购物车中的商品进行添加、修改、删除和更新操作，也可以选择进行结算。其运行的结果如图 22.31 所示。



图 22.31 购物车的运行结果



购物车功能实现的第一步是为想要购买产品的用户分配一辆购物车，使其能够记录自己已经选购的产品。其工作的原理与超市中顾客使用购物车进行购物是相同的，只是这里使用的不是真正意义上的购物车，而是两个 session 变量，一个存储用户选购商品的 ID（$goodsid），另一个存储用户选购该商品的数量（$goodsnum）。如果用户在一次购物中选购多种不同类的商品，则使用“@”对不同类商品的不同 ID 和数量进行分隔。例如，用户选购的不同类商品 id 为 1、2、3，则 session 变量$goodsid 中存储的值为“1@2@3@”；其中同一种商品不能购买两次，如果想要购买多个同种产品，可以在购物车中更改购买商品的数量。在本案例中，购物车的分配功能通过 shopping_cart_first.php 文件来完成，shopping_cart_first.php 文件的代码如下。

代码位置：光盘\TM\sl\22\bcty365\shopping_cart_first.php

```
    <?php session_start();                                              //初始化 session 变量
    session_register("goodsid");                                        //创建一个 session 变量
    session_register("goodsnum");                                       //创建一个 session 变量
if($_SESSION["goodsid"]=="" && $_SESSION["goodsnum"]==""){              //判断 session 变量中的值是否为空
      $_SESSION["goodsid"]=$_GET["id"]."@";                             //如果为空则将商品的 ID 赋给变量
      $_SESSION["goodsnum"]="1@";                                       //将商品数量设置为 1@
}else{
        $array=explode("@",$_SESSION["goodsid"]);                       //如果不为空，则使用@分隔不同的商品 ID
      //判断如果获取的 ID 在 session 变量中已经存在，则提示该商品已经被放入购物车
      if(in_array($_GET["id"],$array)){
          echo "<script>alert('该编程词典已经被放入购物车！');history.back();</script>";
          exit;
      }
     $_SESSION["goodsid"].=$_GET["id"]."@";                             //为 session 变量赋值
     $_SESSION["goodsnum"].="1@";                                       //为 session 变量赋值
}
//将商品放入购物车中，并跳转到购物车页
echo "<script>window.location.href='shopping_cart.php';</script>";
?>
```

在实现购物车的分配和添加商品的功能后，接下来要做的就是查看购物车中的商品，即实现购物车中商品展示的功能。在购物车的商品展示中，可以实现很多的操作，如清空购物车、删除购买商品、更改购买商品数量、继续购物和结算。

购物车商品展示的功能主要通过 shopping_cart.php 文件来完成，首先从 session 变量中读取商品的 ID 和数量，然后根据商品的 ID 循环输出购物车中的商品，最后以商品 ID 为标识符设置不同的超级链接，执行删除商品或者更改购买商品数量等操作。其代码如下。

代码位置：光盘\TM\sl\22\bcty365\shopping_cart.php

```
<table width="720" height="47" border="0" align="center" cellpadding="0" cellspacing="1" bgcolor="#999999">
<?php
      $array=explode("@",$_SESSION["goodsid"]);                 //读取 session 变量中的商品 ID，以@进行分隔
      $arraynum=explode("@",$_SESSION["goodsnum"]);             //读取 session 变量中的商品数量，以@进行分隔
      $markid=0;                                                //创建变量，初始值为 0
      for($i=0;$i<count($array);$i++){                          //应用 for 循环语句循环输出商品 ID 的值
            if($array[$i]!=""){                                 //判断如果商品 ID 的值不为
                  $markid++;                                    //增加变量$markid 的值
```

```
            }
        }
        if($markid==0){                                  //判断如果变量$markid 的值为空则输出下面的内
容
?>
        <tr>
        <td height="22" colspan="4" bgcolor="#FFFFFF"><div align="center">对不起您的购物车中暂无商品信
息!</div></td>
       </tr>
<?php
        }else{                                           //如果$markid 的值不为空，则执行下面的内容
            $totalprice=0;                               //创建变量$totalprice，初始值为 0
            for($i=0;$i<count($array);$i++){             //循环输出数组中的商品 ID 值
                if($array[$i]!=""){
                    //根据获取的商品 ID 的值，从数据库中获取对应产品的信息
                    $sqlcart=mysql_query("select * from tb_bccd where id='".$array[$i]."'",$conn);
                    $infocart=mysql_fetch_array($sqlcart)
?>
        <tr>
            <form name="form<?php echo $array[$i]?>" method="post" action="changegoodsnum.php">
          <td height="22" bgcolor="#FFFFFF"> <?php echo unhtml($infocart["bccdname"]);?></td>
          <td height="22" bgcolor="#FFFFFF"><div align="center"><?php echo
number_format($infocart["price"],2);?></div></td>
          <td height="22" bgcolor="#FFFFFF"><div align="center"><input type="text" name="goodsnum"
value="<?php echo $arraynum["$i"];?>" class="inputcss" size="8" ><input type="hidden" name="id"
value="<?php echo
$infocart["id"];?>" ></div></td>
          <td height="22" bgcolor="#FFFFFF"><div align="center"><a href="javascript:form<?php echo
$array[$i]?>.submit();" class="a1"> 更 改 数 量 </a> | <a href="delgoods.php?id=<?php echo
$infocart["id"];?>"
class="a1">删除该项</a></div></td>
          </form>
        </tr>
<?php
                    $totalprice+=$infocart["price"]*$arraynum["$i"];
                }
            }
        }
?>
</table>
```

22.10.4　商品订单的实现过程

在确定所要购买的商品之后，接下来要做的就是进行购物结算，填写用户购物订单，将订单保存到数据库中，并且随机生成一个订单号，作为订单的唯一标识。生成订单的运行结果如图 22.32 所示。





图 22.32　生成订单的运行结果

订单提交以后用户可以选择汇款的方式：一是选择网上支付，那么将跳转到企业指定的网上银行进行汇款，汇款的操作将在企业指定的网上银行中进行，具体实现过程将在 22.10.5 节进行介绍；二是选择到指定的银行向企业提供的账号中汇款。企业将在收到汇款后按照用户指定的地址和方式将产品送到。商品订单的生成和处理由 shopping_cart_getuserinfo.php 和 savebuyuser.php 文件来完成。

订单处理由 savebuyuser.php 完成，首先连接数据库，随机生成一个订单号，然后获取购物车中的商品信息，最后将商品信息和订单号存储到数据库中。其代码如下。

代码位置：光盘\TM\sl\22\bcty365\savebuyuser.php

```
<?php    session_start();                                                //初始化 session 变量
include_once("conn/conn.php");                                            //连接数据库
$ddnumber=substr(date("YmdHis"),2,8).mt_rand(100000,999999);              //随机生成订单号
$sql=mysql_query("select * from tb_city where id='".$_POST["city"]."'",$conn);   //读取数据库中的城市信息
$info=mysql_fetch_array($sql);
if($shfs=="1"){                                                           //判断用户选择的送货方式
     $yprice=$info[pt];
     $shfs="普通邮递";
}elseif($shfs=="2"){
     $yprice=$info[kd];
     $shfs="邮政特快专递 EMS";
}
$array=explode("@",$_SESSION["goodsid"]);                 //以@来分隔 session 变量中存储的商品 ID
$arraynum=explode("@",$_SESSION["goodsnum"]);             //以@来分隔 session 变量中存储的商品数量
$totalprice=0;
for($i=0;$i<count($array);$i++){                          //循环读取数组中商品的 ID
     if($array[$i]!=""){
          $sqlcart=mysql_query("select * from tb_bccd where id='".$array[$i]."'",$conn);
          $infocart=mysql_fetch_array($sqlcart);
          $totalprice+=$infocart["price"]*$arraynum["$i"];
     }
}
$totalprice=$totalprice+$yprice;                          //获取汇款金额
//将表单中提交的数据存储到数据库中
```

```
if(mysql_query("insert into
tb_dd(ddnumber,recuser,sex,address,yb,qq,email,mtel,gtel,shfs,spc,slc,yprice,totalprice,createtime,cityid)
values('".$ddnumber."','".$_POST["recuser"]."','".$_POST["sex"]."','".$_POST["address"]."','".$_POST["yb"]."','".$
_POST["qq"]."','".$_POST["email"]."','".$_POST["mtel"]."','".$_POST["gtel"]."','".$shfs."','".$_SESSION["goodsid"]
."','".$_SESSION["goodsnum"]."','".$yprice."','".$totalprice."','".date("Y-m-d
H:i:s")."','".$_POST["city"]."')",$conn)){
    session_unregister("goodsid");                              //注销 session 变量 goodsid
    session_unregister("goodsnum");                             //注销 session 变量 goodsnum
    echo "<script>window.location.href='shopping_dd.php?ddno=".base64_encode($ddnumber)."';</script>";
}else{
    echo "<script>alert('订单信息保存失败，请重试!');</script>";
}
?>
```

22.10.5　在线支付的实现

所谓在线支付就是客户端（金融机构需客户端安装由金融机构签发的数字证书，信用卡免安装）将支付信息加密后通过互联网传送到支付网关（支付网关是解决网络上安全支付问题的交易平台，位于互联网和传统的金融机构内部网之间，其主要作用是将互联网和金融网络安全地连接起来，将不安全的网上交易信息传给安全的金融网络，起到隔离和保护金融网络的作用），同时金融机构网上支付系统反馈有关支付信息，客户确认无误后进行支付确定，支付网关负责商户网上交易资金的清算，并根据商户提供的开户行、账号等结账信息将网上消费款项汇总划入商户账户。

BCTY365 网上社区的在线支付是与中国工商银行合作来共同完成的。BCTY365 网上社区的在线支付操作步骤如下。

（1）登录网上社区，如图 22.33 所示。



图 22.33　在线订购的操作页面

（2）购买商品。在本页中，不但可以购买商品，还可以查看商品的详细信息和购物车中的商品信息，如图 22.34 所示。





图 22.34　购买商品操作页面

（3）进入购物车操作页面。在该页面中，可以修改购物数量，删除指定商品，清空购物车，继续购物和统计购买商品的金额，也可以单击“结算”按钮进入到商品结算页面，如图 22.35 所示。

商品名称	单价（元）	数量（个）	操作
VB编程词典	2,000.00	1	更改数量 | 删除该项

<< 继续购买　清空购物车 >>　　商品金额总计：2,000.00 元　　￥结算

图 22.35　购物车操作页面

（4）进入到购物结算页面，填写收货人的详细信息，确认后提交该数据，如图 22.36 所示。



图 22.36　填写收货人的详细信息

（5）订单确认。订单确认以后，就可以提交订单，准备进行网上支付，如图 22.37 所示。



图 22.37 订单确认

（6）进行网上支付。在这里可以选择工行网上支付，也可以选择取消该订单，如图 22.38 所示。

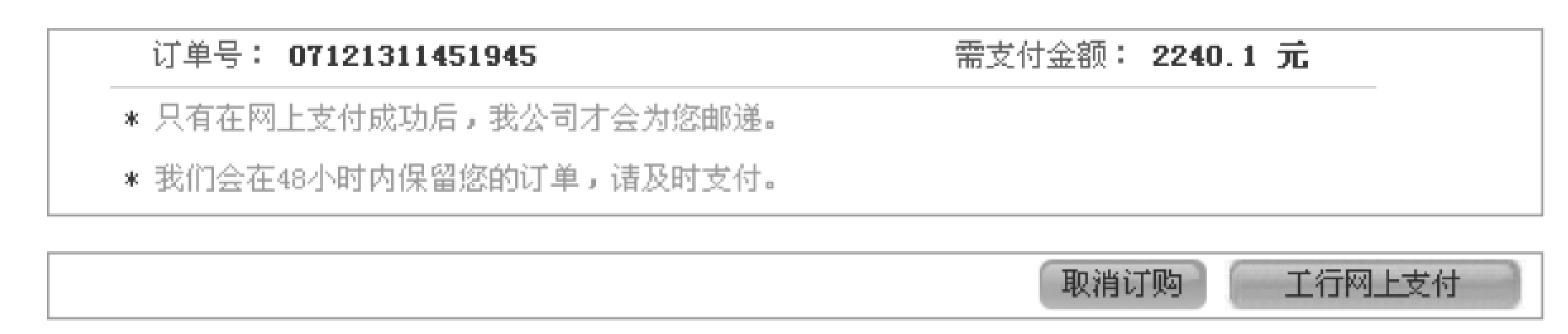


图 22.38 执行网上支付

接下来的操作在工行 B2C 支付页面上进行。首先网上社区按照工商银行 B2C 订单数据规范形成提交数据，并使用工商银行提供的 API 和商户证书对订单数据签名，形成 form 表单返回客户浏览器，表单 action 地址指向工商银行接收商户 B2C 订单信息的 servlet；然后在客户确认使用工行网上支付后，提交此表单到工商银行；最后工行网银系统接收此笔 B2C 订单，对订单信息和商户信息进行检查，通过检查则显示工行 B2C 支付页面。

客户通过工行 B2C 支付页面实现网上支付，商户查询网上银行的账户，如果货款已经到账，则根据客户指定的方式将货物送达客户手中。

上述内容就是网上社区系统的在线支付流程，涉及工商银行的操作内容这里不做讲解。这里主要讲解一下如何将订单信息提交到工商银行。该项操作主要通过 shopping_tjdd.php 文件来实现，首先从数据库中读取订单信息，然后将订单信息进行输出，最后创建“取消订购”和“工行网上支付”两个超链接，通过 JavaScript 脚本来调用不同的执行该文件。其关键代码如下。

```
<?php    include_once("conn/conn.php");    include_once("top.php");  //连接数据库和网站的头文件  ?>
<!-->省略了部分代码<!-->
<?php
$ddnumber=base64_decode($_GET["ddno"]);                          //对获取的订单编号进行 base64 解码
//获取该订单的金额信息
$sql=mysql_query("select * from tb_dd where ddnumber='".$ddnumber."'",$conn);
$info=mysql_fetch_array($sql);
$amount=$info["totalprice"];
$amount=str_replace(",","",number_format($amount,2));             //修改数字的输出格式
$amount=str_replace(".","",number_format($amount,2));             //修改数字的输出格式
```

```
?>
…                                                    //省略部分 HTML 代码
<table width="630" border="0" align="center" cellpadding="0" cellspacing="0">
      <tr><td width="159"> 
<?php
      $sql=mysql_query("select totalprice from tb_dd where ddnumber='".base64_decode($_GET["ddno"])."'",
$conn);
      $info=mysql_fetch_array($sql);
            echo "<font color=red><strong>".$info["totalprice"]." 元</strong></font>";
?>          </td>
      </tr>
</table>
…                                                    //省略部分 HTML 代码
<script language="javascript">
//打印订单
function openprintwindow(x,y,z){
      window.open("printwindow.php?ddno="+x+"&pv="+z,"newframe","top=200,left=200,width=635,hei
ght="+(230+20*y)+",menubar=no,location=no,toolbar=no,scrollbars=no,status=no");
       }
</script>
…                                                    //省略部分 HTML 代码
<table width="630" height="25" border="0" align="center" cellpadding="0" cellspacing="0">
      <tr>
           <!-- ---------------------------------------------取消订单------------------------------------------- -->
            <td width="75"><img src="images/bg_14(14).jpg" width="69" height="20" style="cursor:hand"
onclick="javascript:if(window.confirm('如果取消该订单，则该订单将被删除，您需要重新购买！
')==true){window.location.href='deletedd.php?ddno=<?php echo $_GET["ddno"];?>';}"/></td>
           <!-- --------------------------------------------执行网上支付------------------------------------------ -->
            <td width="125"><img src="images/bg_14(15).jpg" width="119" height="20"
onclick="javascript:window.location.href='ddform.php?orderid=<?php echo
base64_decode($_GET["ddno"]);?>&amount=<?php       echo       $amount;?>&orderDate=<?php       echo
date("Ymdhis");?>';"
style="cursor:hand"/></td>
      </tr>
</table>
<!-- ------------------------------------------------------省略了部分代码------------------------------------------- -->
<?php      include_once("bottom.php");                    //包含网站的尾文件            ?>
```

有关在线支付流程中的其他操作实现方式已经在 22.10 节中进行了详细的讲解，这里不再赘述。其具体的代码还可以参考本书光盘中的 TM\02\bcty365\文件。

22.11　社区论坛模块设计

社区论坛模块为网站的浏览者提供一个交流的平台，以此来扩大网站的影响力，汇聚更多的人气，宣传企业形象，推广企业产品。

22.11.1　社区论坛模块概述

社区论坛模块为浏览者、会员、客户和企业之间提供一个大的交流平台，根据身份的不同，给予其分别拥有不同的操作权限，社区论坛模块的操作流程如图 22.39 所示。

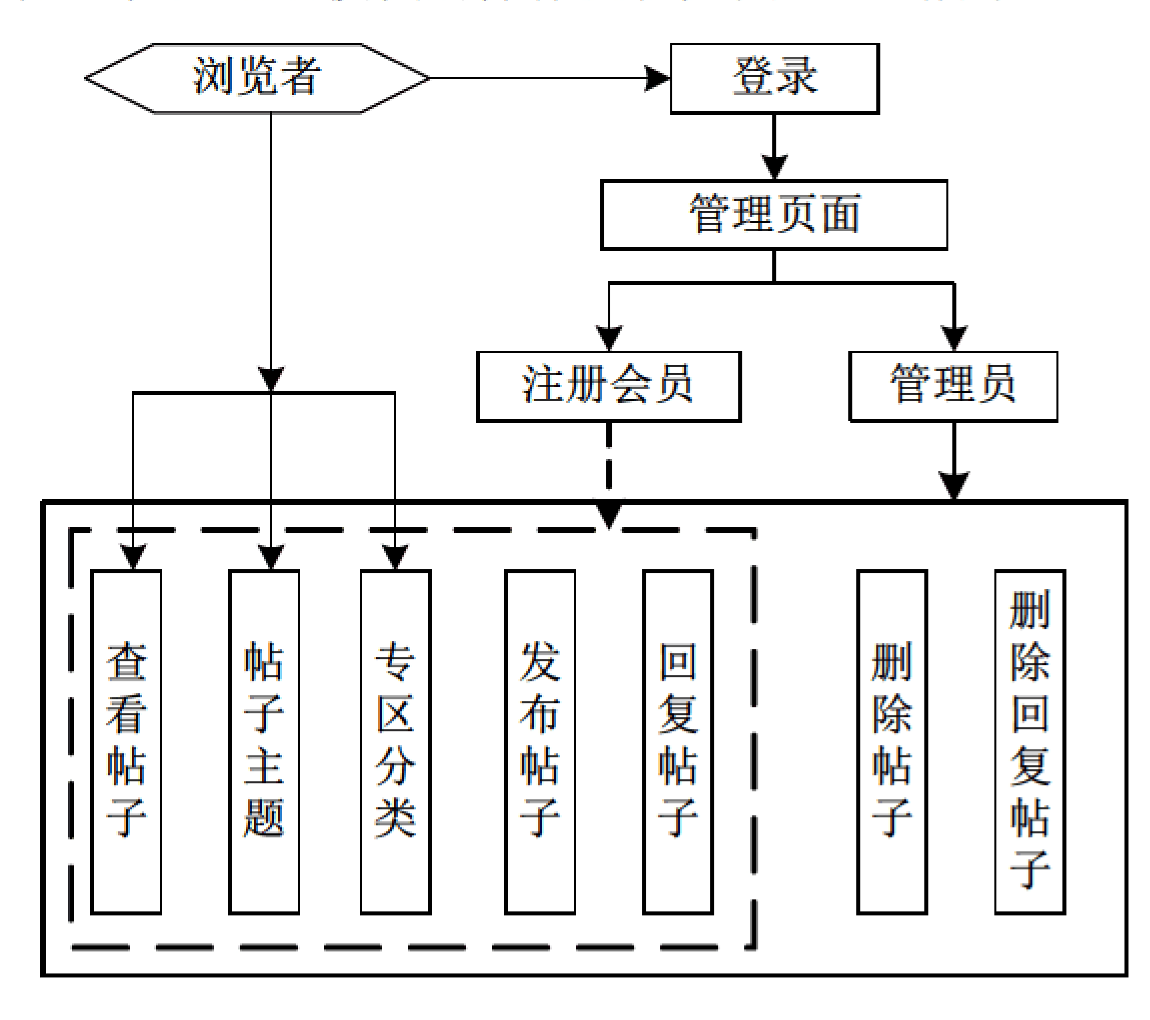


图 22.39　社区论坛流程图

在本论坛中，浏览者只能够查看帖子；注册会员既可以查看帖子，也可以发布和回复帖子；管理员则具有发布、回复、查看和删除的权限。

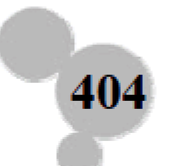
22.11.2　社区论坛模块技术分析

在社区论坛模块的实现过程中，通过 JavaScript 脚本和下拉列表框的结合实现一个不同版块之间快速跳转的功能，从而能够更加灵活、方便地实现不同版块之间的跳转。

下面分析该技术是如何实现的，该技术的实现综合 3 个方面的内容，以一个下拉列表框为主，通过 PHP 语句从数据库中读取数据作为下拉列表框的值，应用 onchange 事件来调用 JavaScript 脚本，实现不同版块之间的跳转。这里以 bbs_top.php 文件中的快速跳转功能为例进行分析。其关键代码如下。

代码位置：光盘\TM\sl\22\bcty365\bbs_top.php

```
<!--创建一个下拉列表框，指定名称为 select_type，并且设置其属性，通过 onchange 事件来调用 JavaScript 脚本文件，实现页面跳转-->
<select name="select_type" class="inputcss"
     onChange="javascript:window.location=this.options[this.selectedIndex].value;" >
<?php
//通过 PHP 语句从数据库中读取数据，使用数据的 ID 作为下拉列表框的值，使用数据的标题 title 作为下拉列表框显示的内容
     $sql=mysql_query("select * from tb_type_small order by createtime desc",$conn);
```



```
        $info=mysql_fetch_array($sql);
        if($info==""){
                echo "<option>暂无讨论区</option>";
        }else{
                echo "<option>-版块快速跳转-</option>";
                do{                                             //应用 do…while 循环语句输出下拉列表框中的值
                echo "<option value='bbs_list.php?id=".$info[id]."'>".$info[title]."</option>";
                }
                while($info=mysql_fetch_array($sql));           //应用 do…while 循环语句结束
        }
?>
</select>
```

该技术实现的运行结果如图 22.40 所示，它将实现从硬件查询模块跳转到 PHP 模块。

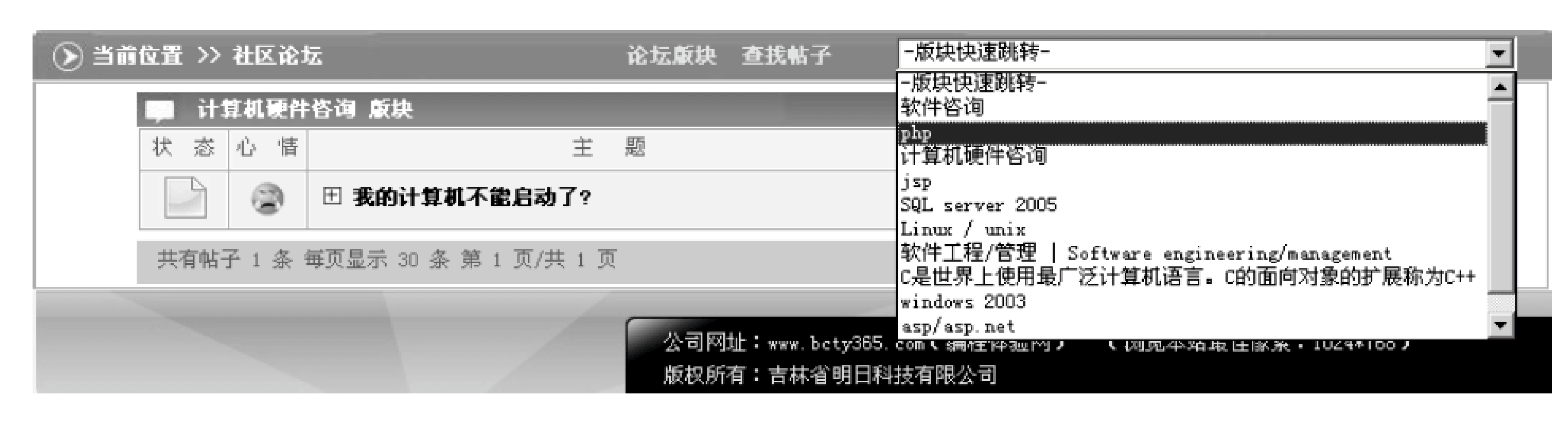


图 22.40　版块跳转功能的运行结果

22.11.3　论坛分类的实现过程

论坛分类可以分为两类，一个是论坛中大的版块分区，分为 6 个版块：综合信息讨论区、操作系统、程序设计交流区、网管技术应用、Web 程序开发和数据库技术，其数据存储于 tb_type_big 表中。另一个是对应不同的版块中不同语言和技术的分类，分为 11 种，其数据存储于 tb_type_small 表中。论坛分类的运行结果如图 22.41 所示。

Web程序开发	发帖数	热门帖数
计算机硬件咨询 创建时间：2008-12-03 00:00:00	1	0
操作系统	**发帖数**	**热门帖数**
asp/asp.net 创建时间：2008-12-16 00:00:00	0	0
php 创建时间：2009-12-05 00:00:00	1	0
综合信息讨论区	**发帖数**	**热门帖数**
C是世界上使用最广泛计算机语言。C的面向对象的扩展称为C++ 创建时间：2005-12-06 00:00:00	0	0
软件工程/管理 | Software engineering/management 创建时间：2006-12-05 00:00:00	0	0

图 22.41　论坛分类的运行结果

论坛分类的实现原理很简单，首先从 tb_type_big 表中读取 6 个版块中的数据，进行循环输出，然后在版块中嵌套循环，用于输出不用语言的分类数据。该功能主要通过 bbs_index.php 文件来完成，bbs_index.php 文件的程序代码如下。

代码位置：光盘\mr\sl\22\bcty365\bbs_index.php

```
<?php    include_once("top.php");                              //调用网站头文件
         include_once("bbs_top.php");                          //调用社区论坛的头文件
?>
<?php
//循环输出数据表 tb_type_big 中的 6 个版块数据
$sql=mysql_query("select * from tb_type_big order by createtime desc",$conn);
$info=mysql_fetch_array($sql);
if($info==false){                                              //如果返回值为 false，则执行下面的内容
?>
...                                                            //省略了部分代码
<?php
     }else{                                                    //如果返回值为 true，则执行 do…while 循环语句
         /* 外部嵌套循环，输出论坛中的版块分类数据 */
         do{
?>
…省略了部分代码
<table width="750" border="0" align="center" cellpadding="0" cellspacing="1" bordercolor="#FFFFFF"
bgcolor="#6EBEC7">
<?php
     //循环输出 tb_type_small 表中的不同语言和技术的分类数据
     $sql1=mysql_query("select * from tb_type_small where bigtypeid='".$info["id"]."'",$conn);
     $info1=mysql_fetch_array($sql1);
     if($info1==false){                                        //判断如果返回值为 false，则执行下面的内容
?>
…省略了部分代码
<?php
      }else{        //如果返回值为 true，则执行下面的内容，输出该版块中对应语言和技术的帖子详细信息
?>
…                                                              //省略了部分代码
     <?php    /* 内部嵌套循环，输出不同语言和技术的分类 */
              do{         ?>
         <tr>
              <td height="30"><font color="#666666">创建时间：<?php echo $info1["createtime"];?></font>
</td>
            </tr>
…省略了部分代码
        <?php
              }while($info1=mysql_fetch_array($sql1));
              /* 内部嵌套循环结束 */
      }
       ?>
</table>
  <?php
         }while($info=mysql_fetch_array($sql));
```

```
            /* 外部嵌套循环结束，对版块中的大类进行输出 */
}
 ?>
<?php      include_once("bottom.php");          ?>
```



注意

在应用 do…while 循环语句时，while 后的分号不能省略。

22.11.4　论坛帖子浏览的实现过程

论坛帖子浏览主要输出指定帖子的详细信息，包括发帖人、用户级别和注册的时间，以及帖子的主题、内容和发帖时间，包括上传的图片。本模块中是用户权限使用体现得最明显的地方，可以分为 3 种情况：第一以浏览者进行登录，只能是浏览帖子的内容，没有其他权限；第二以会员进行登录，可以对帖子进行回复，发表自己的看法；第三以管理员的身份进行登录，不但可以回复帖子，而且可以对任何人发布和回复的帖子进行删除和顶帖的操作。下面就来看一下以管理员身份进行登录时都具备哪些权限，运行结果如图 22.42 所示。



图 22.42　管理员浏览帖子的结果

论坛帖子浏览的功能通过 bbs_looks.php 文件完成，首先根据传递的 ID 值读取指定的帖子数据，然后判断登录用户的类型，最后根据用户不同的类型执行不同的操作。其代码如下。

代码位置：光盘\TM\sl\22\bcty365\bbs_lookbbs.php

```
<?php
```

```
//根据$_GET 传递的数据获取 tb_bbs 中的数据
$sqlb=mysql_query("select * from tb_bbs where id='".$_GET["id"]."'",$conn);
$infob=mysql_fetch_array($sqlb);
//根据$_GET 传递的数据获取 tb_user 中的数据
$sql4=mysql_query("select * from tb_user where id='".$infob["userid"]."'",$conn);
$info4=mysql_fetch_array($sql4);
?>
…                                                                    //省略了部分 HTML 代码
<table width="180" border="0" align="center" cellpadding="0" cellspacing="0">
    <tr>
    <td height="22">用户级别：
            <?php
                //根据用户信息表 tb_user 中字段 usertype 的值判断该用户的类型
                //如果值为 1 则是管理员，值为 2 则是后台管理员，值为 0 则是普通会员
                if($info4["usertype"]=="1") echo "管理员";else echo "普通会员";
            ?>
        </td>
    </tr>
</table>
…                                                                    //省略了部分 HTML 代码
<table width="500" height="200" border="0" align="center" cellpadding="0" cellspacing="0">
    <tr>
    <td width="35" height="20"><div align="center"><img src="images/lt_15(11).jpg" width="25"
height="25"></div></td>
       <td><?php echo $infob["createtime"];?></td>
    </tr>
    <tr>
    <td height="150" colspan="2">
            <?php
//判断 tb_bbs 表中的字段 photo 是否为空，为空则执行下面的内容
            if($infob[photo]!=""){
                $photos=substr($infob[photo],2,70);          //获取图片在服务器中的存储路径
                    echo (stripslashes($infob["content"])); //输出帖子的内容
                 echo "<img src=\"$photos\">";              //根据获取的图片路径，输出服务器中的图片
              }else{                              //如果 tb_bbs 表中的图片字段 photo 为空，则执行下面的内容
                    echo (stripslashes($infob["content"])); //只输出帖子的内容
              }
            ?>
        </td>
    </tr>
</table>
<table width="530" height="20" border="0" align="center" cellpadding="0" cellspacing="0">
    <tr>
        <td width="239" height="30"> </td>
       <td width="291"><img src="images/lt_15(5).jpg" width="72" height="23" style="cursor:hand" onclick="
            <?php
                //如果$_SESSION["unc"]的值为空，则不可以进行顶帖子的操作
                if($_SESSION["unc"]==""){
                    echo  "javascript:alert('请先登录本站，然后进行此操作！');window.location.href=
```

```
'index.php';";
                        }else{
                            //否则将判断当前用户的类型，如果是管理员则可以顶帖
                            $sqlu=mysql_query("select usertype from tb_user where
usernc='".$_SESSION["unc"]."'",$conn);
                            $infou=mysql_fetch_array($sqlu);
                            if($infou["usertype"]==1){                 //如果用户的类型为 1，则有顶帖的权限
                                echo "javascript:window.location.href='settop.php?id=".$infob["id"]."'";
                            }else{                                     //否则不具备该权限
                                echo "javascript:alert('对不起，您不具备该操作权限！');";
                            }
                        }
                    ?>
                    "/>  
                    <?php
                        //判断当前用户是否具有删除帖子的权限
                        if($_SESSION["unc"]!=""){
                            //条件为用户不能为空，并且是管理员，才具备删除帖子的权限
                            $sqlu=mysql_query("select usertype from tb_user where usernc='".$_SESSION
["unc"]."'",$conn);
                        $infou=mysql_fetch_array($sqlu);
                            if($infou["usertype"]==1){
                    ?>
                    <img src="images/lt_15(10).jpg" onclick="javascript:if(window.confirm('您确定删除该帖吗？
')==true){window.location.href='bbs_delete.php?id=<?php echo $infob["id"]?>';}" style="cursor:hand"/>
                    <?php
                            }
                        }
                    ?>
                </td>
            </tr>
</table>
```

说明

上面给出的是该文件的部分代码，主要讲解了该功能的实现方法，完整的代码可以参考本书光盘中的 TM\02\bcty365\bbs_lookbbs.php 文件。

22.11.5　论坛帖子发布的实现过程

论坛帖子发布通过两个文件来完成，一个是帖子发布信息的提交页 bbs_pubs.php，另一个是对提交的数据进行处理的 retrieve.php 文件。该功能实现的运行结果如图 22.43 所示。

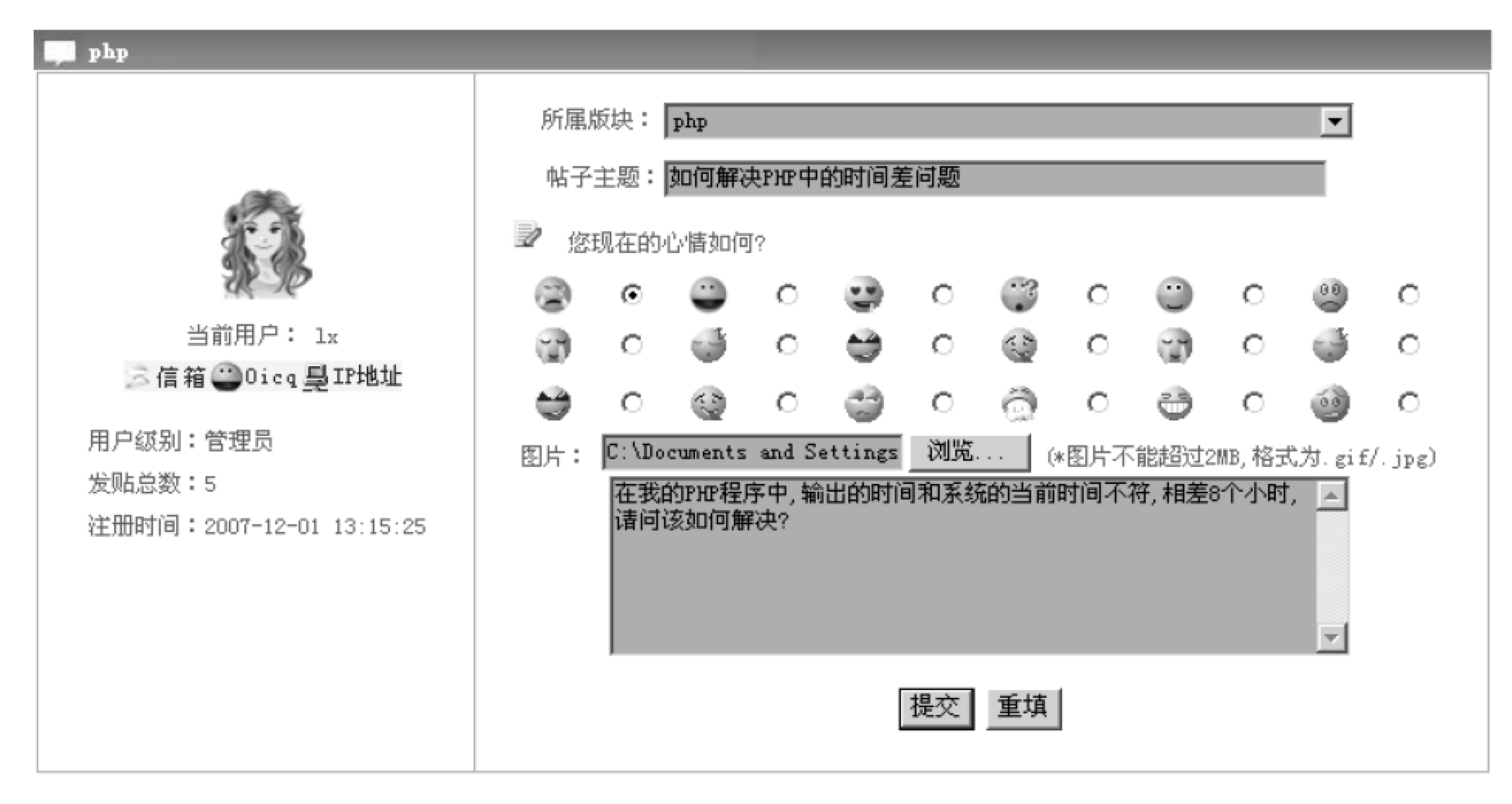


图 22.43　帖子发布模块的运行结果

在发布信息的提交页中，显示当前用户的个人信息，设置添加数据表单元素，其中表单元素的设计如表 22.3 所示。

表 22.3　发布信息页中使用的表单元素

名　称	元素类型	重要属性	含　义
form_bbs	form	method="post" action="retrieve.php" enctype="multipart/form-data"	发帖表单

bbs_type	select	class="inputcss" style="background-color:#6EBEC7">
		<?php
		$sql=mysql_query("select * from tb_type_small order by createtime desc",$conn);
		$info=mysql_fetch_array($sql);
		if($info==false){
		echo "<option>暂无讨论区</option>";
		}else{
		do{
		?>
		<option value="<?php echo $info[id] ;?>"<?php if($_GET[id]==$info[id]) {echo "selected=\"selected\"";}?>><?php echo $info[title];?></option>
		<?php }
		while($info=mysql_fetch_array($sql));
		}?>
			选择发表帖子的语言或者技术的类别

bbs_title	text	class="inputcss" style="background-color:#6EBEC7">	帖子标题
bbs_head	radio	value="<?php echo("images/bbsface/face".($i-1).".gif");?>"	表情图
bbs_photo	file	id="bbs_photo" class="inputcss" style="background-color:#6EBEC7" />	上传图片
content1	textarea	id="content1" class="inputcss" style="background-color:#6EBEC7">	帖子内容
Submit	submit	value="提交"	提交表单

在 retrieve.php 页中对表单提交的数据进行处理，将数据存储到 tb_bbs 表中，并且更新用户信息表 tb_user 中 pubtimes 字段的值，其中还应用了图片上传技术，将图片上传到服务器中指定的文件夹下。retrieve.php 文件的代码如下。

代码位置：光盘\TM\sl\22\bcty365\retrieve.php

```
<?php    session_start();                                              //初始化 session 变量
$title=$_POST[bbs_title];                                              //获取帖子的标题
$content=$_POST[content1];                                             //获取帖子的内容
/* 判断提交的帖子主题和帖子内容是否为空*/
if($title==""){
    echo "<script>alert('请输入帖子主题!');history.back();</script>";
    exit; }
if($content==""){
    echo "<script>alert('请输入帖子内容!');history.back();</script>";
    exit; }
/*********/
include_once("conn/conn.php");                                         //连接数据库
//根据$_SESSION["unc"]的值读取数据库中用户的信息
$sql=mysql_query("select * from tb_user where usernc='".$_SESSION["unc"]."'",$conn);
$info=mysql_fetch_array($sql);                                         //检索指定条件的数据信息
$userid=$info[id];                                                     //获取用户 id
$typeid=$_POST[bbs_type];                                              //接收版块名称
$title=$_POST[bbs_title];                                              //接收帖子主题
$content=$_POST[content1];                                             //接收帖子内容
$head=$_POST[bbs_head];                                                //接收头像
$createtime=date("Y-m-j H:i:s");                                       //获取系统当前时间
$lastreplytime=$createtime;                                            //将当前时间赋给变量
$readtimes=0;
$link=date("YmjHis");
  if($_FILES['bbs_photo']["name"]==true){       //上传图片，判断文件是否存在，如果存在则执行下面的内容
    $photo_name=strtolower(stristr($_FILES["bbs_photo"]["name"],"."));//获取图片后缀名，将字符转成小写
    if($photo_name!=".gif" & $photo_name!=".jpg" & $photo_name!=".jpeg" ){ //判断图片的格式是否符合要求
    echo "<script>alert('您上传的图片格式不正确!');history.back();</script>";
    }else{
        $paths1=$link.mt_rand(1000000,9999999).$photo_name;     //创建图片的名称
    $photos="./upfile/".$paths1;                                //创建图片的存储路径
            move_uploaded_file($_FILES['bbs_photo']["tmp_name"],$photos);
                                                                //将图片存储到指定的文件夹下
        //向数据库添加数据
        if(mysql_query("insert into tb_bbs(userid,typeid,title,content,createtime,lastreplytime,head,readtimes,
top,photo)
values('".$userid."','".$typeid."','".$title."','".$content."','".$createtime."','".$lastreplytime."','".$head."','".$readtimes
."','0','$photos')",$conn)){
            mysql_query("update tb_user set pubtimes=pubtimes+1",$conn);
                                                                //更新 tb_user 中 pubtimes 字段的值
            echo "<script>alert('新帖发表成功!');history.back();</script>";
        }else{
```

```
                echo "<script>alert('新帖发表失败!');history.back();</script>";
            }
        }
}else{//如果没有提交图片，则执行下面的内容
        if(mysql_query("insert into tb_bbs(userid,typeid,title,content,createtime,lastreplytime,head,readtimes,top)
values
('".$userid."','".$typeid."','".$title."','".$content."','".$createtime."','".$lastreplytime."','".$head."','".$readtimes."','0')",
$conn)){
            mysql_query("update tb_user set pubtimes=pubtimes+1",$conn);
            echo "<script>alert('新帖发表成功!');history.back();</script>";
        }else{
            echo "<script>alert('新帖发表失败!');history.back();</script>";
        }
}
?>
```

22.11.6 论坛帖子回复的实现过程

回复论坛中的帖子，必须以会员或者管理员的身份进行登录，否则不能进行帖子的回复操作，其运行结果如图 22.44 所示。



图 22.44 论坛帖子回复的运行结果

论坛帖子回复功能的实现主要通过 bbs_looks.php 和 savereply.php 两个文件。其中应用 JavaScript 脚本对回复帖子的文本框进行输出和隐藏的控制。在 bbs_looks.php 文件中，帖子回复使用的表单元素如表 22.4 所示。

表 22.4　论坛帖子回复中的重要表单元素

名　称	元 素 类 型	重 要 属 性	含　义
form_reply	form	method="post" action="savereply.php" enctype="multipart/form-data">	回复表单
reply_title	text	class="inputcss" id="reply_title"	回复帖子主题
bbsid	hidden	value="<?php echo $infob["id"];?>"	对应帖子的 ID
bbs_head	radio	value="<?php echo("images/bbsface/face".($i-1).".gif");?>"	表情图
bbs_photo	file	id="bbs_photo" class="inputcss"	上传图片
content1	textarea	id="content1"	回复帖子内容
Submit	submit	value="提交"	提交表单

在帖子回复表单 bbs_looks.php 页中，首先判断登录用户是否具有回复的权限，然后根据提交的值展开回复表单的文本框，在文本框中输入回复的主题和内容，最后将数据提交到表单处理页 savereply.php 中。bbs_looks.php 的主要代码如下。

代码位置：光盘\TM\sl\22\bcty365\ bbs_lookbbs.php

```
<script language="javascript">
//设计回复帖子表格的输出方式
function show_reply(){                                          //定义一个函数
	if(reply_bbs1.style.display=="")                        //判断当 display 的值为空时
		{
			reply_bbs1.style.display="none" ;               //则输出表格
			button_show_bbs.value="回复帖子";                //显示回复帖子
		}
	else if(reply_bbs1.style.display=="none")               //判断当 display 的值为 none 时
		{
			reply_bbs1.style.display="" ;                   //则不输出表格
			button_show_bbs.value="关闭窗口";                //显示关闭窗口
		 }
}
</script>
<!-- -------------------------------------判断登录用户是否具有回复的权限----------------------------------------- -->
<img src="images/lt_15(9).jpg" width="72" height="23" id="button_show_bbs"   style="cursor:hand" onClick="
<?php
	if($_SESSION["unc"]==""){
		echo "javascript:alert('请先登录本站，然后回复帖子!');window.location.href='index.php';";
	}else{
?>
	show_reply()
<?php     }     ?>          "/>
```

表单处理页 savereply.php 将表单提交的数据存储到指定的数据库中，其实现的方法与论坛发布中的表单处理技术是相同的，有关该技术的详细讲解请参考 22.11.5 节，这里不再赘述。

22.12 后台首页设计

作为一个完整的网上社区系统，要想能够及时地对网站进行管理和维护，必须具有一个强大的后台管理系统，对网上社区系统中的数据进行更新和维护。

22.12.1 后台首页概述

网上社区系统的后台管理采用的是一种简单的框架结构，通过 switch 语句来实现。其具体内容如下。

（1）软件试用管理：包括软件试用产品的添加和删除。

（2）编程词典管理：包括编程词典版本的添加、删除和编程词典内容的添加和删除。

（3）在线订购管理：主要用于管理用户提交的订单。

（4）软件升级管理：包括升级包的添加、删除和序列号的添加和删除。

（5）站内公告管理：主要用于添加和删除站内公告。

（6）技术支持管理：主要用于添加常见问题和删除常见问题，以及对客户反馈信息进行管理。

下面看一下本案例中提供的后台首页，该页面在本书光盘中的路径为\TM\02\bcty365\admin\index.php，如图 22.45 所示。



图 22.45 BCTY365 网上社区系统后台首页

22.12.2　后台首页技术分析

网上社区后台首页的设计主要应用 switch 语句和 include 语句，其实现的原理是：应用 switch 语句，根据超链接中传递的变量值进行判断，根据不同的变量值应用 include 调用不同的子文件。该技术的实现流程如图 22.46 所示。



图 22.46　网上社区后台首页设计流程

为能够更好地理解这个技术，先来了解一下 switch 语句。该语句的格式如下。

```
switch( expr ){                              //expr 条件为变量名称
      case expr1:                            //case 后的 expr1 为变量的值
            statement1;                      //冒号":"后的是符合该条件时要执行的部分
      break ;                                //应用 break 来跳离循环体
     case expr2 :
           statement2 ;
      break ;
      default:
            statementN;
      break;
}
```

参数 expr 是表达式的值，即 switch 语句的条件变量的名称；参数 expr1 放置于 case 语句之后，是要与条件变量 expr 进行匹配的值中的一个；statement1 是在参数 expr1 的值与条件变量 expr 的值相匹配时执行的代码；break 语句实现终止语句的执行，即当语句在执行过程中，遇到 break 就停止执行，跳出循环体；default 是 case 的一个特例，匹配了任何其他 case 都不匹配的情况，并且是最后一条 case 语句。

通过 switch 和 include 语句来实现后台管理功能的设计是一个很好的方法，不但实现过程简单，而且操作也非常灵活。其关键代码如下：

代码位置：光盘\TM\sl\22\bcty365\ admin\wzdh.php

```
<?php
switch($htgl){                                          //根据变量提交的不同值
    case "添加编程词典版本":                             //判断与变量提交的值是否相同
         include("addbb.php");                          //如果值相同，则调用指定的文件
    break;                                              //并且跳出本次循环
  case "编辑编程词典版本":
  include("editbd.php");
    break;
      …                                                 //部分代码省略
      case "":                                          //当变量的值为空时
  include("edittell.php");                              //调用该文件
    break;
}
?>
```

22.12.3 后台首页的实现过程

在后台首页的设计过程中，以 switch 循环语句为基础，架设整个后台管理功能的框架结构；充分发挥 include 语句的作用，调用不同的文件执行不同的管理操作；应用 JavaScript 脚本来控制栏目列表的输出和隐藏。

控制栏目列表的输出和隐藏在 menu.php 文件中进行，首先定义一个函数 change()用于控制表格的输出和隐藏，然后在表格中应用 onclick 事件传递不同的值到自定义函数 change()，最后根据不同的值显示不同的内容。其关键代码如下。

代码位置：光盘\TM\sl\22\bcty365\admin\menu.php

```
<script language="javascript">
//通过脚本语言控制文本框的伸展和收缩
function change(x,y){                              //定义一个函数
      if(x.style.display=="none"){                 //判断当样式的值为 none 时
          x.style.display="";                      //判断当样式的值为 none 时，输出样式的值为空
      }
      else if(x.style.display==""){                //判断当样式的值为空时
          x.style.display="none";                  //判断当样式的值为空时，输出样式的值为 none
          y.background="images/bg_16_11.jpg";      //输出背景图片
            }
      }
</script>
<table width="175" height="28" border="0" align="center" cellpadding="0" cellspacing="4"
onclick="change(tz1,img_tz1)" style="cursor:hand">
  <tr>
    <td background="images/bg_16_11.jpg" id="img_tz1" class="a4"><div align="left"><img
src="images/bg_16_21.jpg"> 编程词典管理</div></td>
  </tr>
</table>
```

```
<table name="tz1" id="tz1" width="170" height="40" border="0" align="center" cellpadding="0" cellspacing="0"
<?php
     //根据变量的值选择执行的内容，当变量的值不为真时，隐藏该表格
     if(!($_GET[htgl]=="添加编程词典版本" ||$_GET[htgl]=="编辑编程词典版本" ||$_GET[htgl]=="添加编程词典"
||$_GET[htgl]=="编辑编程词典" )){
?>
     style="display:none"
<?php    }   ?>
>
   <tr>
    <td width="40" height="24" background="images/bg_16_16.jpg"> </td>
    <td width="114" background="images/bg_16_16.jpg"><div align="left"><a href="default.php?htgl=添加编程
词典版本">添加编程词典版本</a></div></td>
  </tr>
</table>
```

说明

这里给出的只是后台首页实现过程中的主要代码，详细代码可参考本书的光盘TM\02\bcty365\admin\文件夹下的相关文件。

22.13　编程词典管理模块设计

本模块的功能是对网站中的编程词典进行管理，包括添加编程词典版本、编辑编程词典的版本、添加编程词典和编辑编程词典。

22.13.1　编程词典管理模块概述

本模块的主要功能是管理网站中在线出售的编程词典软件，实现对编程词典软件及时的更新和维护，其管理的内容主要包括添加和编辑编程词典的版本，添加和编辑编程词典的详细信息。在添加编程词典时，包括名称、版权、图片、类别、内容简介和不同版本的共同点；编辑编程词典包括版本、价格、简介、功能和服务，其中每一个编程词典软件只可以编辑一次，不可以进行重复编辑，如果要重新编辑，就必须将已经编辑过的信息删除。

22.13.2　编程词典管理模块技术分析

在编程词典管理模块中，应用到图片上传技术，通过该技术将编程词典的界面效果上传到服务器的指定文件夹下。该技术主要通过 move_uploaded_file()函数来实现，其中还应用到 is_dir()、mkdir()函数，判断指定的文件夹是否存在和创建文件夹，还有 mt_rand()、strstr()函数和$_FILES[]全局变量。为

了更好地理解和掌握图片上传处理技术，这里以编程词典模块中的 savebccd.php 文件为例进行讲解。

首先应用 is_dir()函数判断在服务器中是否存在指定的文件夹，如果不存在，则应用 mkdir()函数创建一个新的文件夹。

然后应用$_FILES[]全局变量获取图片名，应用 strstr()函数获取图片文件的后缀名，为避免出现同名文件覆盖，这里应用系统的当前时间和 mtrand()函数获取的一个 7 位随机数字作为图片的名称。

最后确定图片在服务器中存储的路径，将图片上传到指定的文件夹下。而数据库中存储的数据是图片在服务器中的路径，当需要输出图片时，只需要获取到数据库中图片的路径即可。savebccd.php 文件的代码如下。

代码位置：光盘\TM\sl\22\bcty365\admin\savebccd.php

```
<?php include_once("../conn/conn.php");                    //连接数据库
$bccdname=$_POST[bccdname];                                //获取 POST 方法提交的值
$owner=$_POST[owner];
$typeid=$_POST[typeid];
$content=$_POST[content];
$samepart=$_POST[samepart];
$addtime=date("Y-m-j H:i:s");                              //获取当前时间
   if(is_dir("./bccdimages")==false){                      //判断指定的文件是否存在
     mkdir("./bccdimages");                                //如果不存在，则创建一个新的文件夹
}
$link=date("YmjHis");                                      //获取当前时间
//为表单中提交的数据重新命名，以当前时间和随机数作为名称
//其中使用$_FILES 获取表单中真实的名称，使用 strstr 函数获取文件的后缀
   $path=$link.mt_rand(1000000,9999999).strstr($_FILES["imageaddress"]["name"],".");
$address="./bccdimages/".$path;                                //定义文件上传的路径
   move_uploaded_file($_FILES["imageaddress"]["tmp_name"],$address);//将文件上传到指定的文件中
$imageaddress="./admin/bccdimages/".$path;                     //获取上传文件在服务器中的存储路径
//将表单中提交的数据存储到数据库中
$query=mysql_query("insert into tb_bccd(bccdname,owner,typeid,content,samepart,imageaddress,addtime)
values('$bccdname','$owner','$typeid','$content','$samepart','$imageaddress','$addtime')",$conn);
if($query==true){
     echo "<script>alert('编程词典添加成功！');history.back();</script>";
}else{
     echo "<script>alert('编程词典添加失败！');history.back();</script>";
}
?>
```

注意

应用 POST 方法上传图片文件时，应当在上传表单的<form>标记中添加以下内容 enctype="multipart/form-data"。

22.13.3　添加编程词典的实现过程

添加编程词典的功能是向数据库中添加编程词典的详细信息，包括编程词典的名称、版权、图片、类别、内容简介和不同版本的共同点。其运行结果如图 22.47 所示。

图 22.47　添加编程词典模块的运行结果

添加编程词典信息模块主要通过 addbccd.php 和 savebccd.php 文件来完成，其中在 addbccd.php 文件中主要是设计表单元素，而 savebccd.php 文件主要是对表单中提交的数据进行处理。addbccd.php 文件中使用的表单元素如表 22.5 所示。

表 22.5　添加编程词典页中使用的重要表单元素

名　称	元 素 类 型	重 要 属 性	含　义
form1	form	method="post" action="savebccd.php" onSubmit="return chkinput(this)" enctype="multipart/form-data">	编程词典表单
bccdname	text	size="25" class="txt_grey"	编程词典名称
owner	text	" size="25" class="txt_grey"	版权所有者
imageaddress	file	size="25" class="txt_grey"	界面图片

typeid	select	<?php　include_once("../conn/conn.php");
		$sql=mysql_query("select * from tb_type order by createtime desc",$conn);
		$info=mysql_fetch_array($sql);
		if($info==false){ echo "<option >暂无类别</option>";
		}else{
		do{
		echo "<option value=".$info[id].">".$info[typename]."</option>"; }
		while($info=mysql_fetch_array($sql));}　　?>
			选择编程词典的版本

content	textarea	rows="10" cols="65" class="textarea">	内容简介

续表

名　称	元素类型	重要属性	含　义
samepart	textarea	rows="10" cols="65" class="textarea">	不同版本的特点
Submit	submit	value="添加" class="btn_grey"	提交表单

savebccd.php 文件实现对表单中提交的数据进行处理，首先通过$_POST 获取表单中提交的数据，然后判断指定的文件夹是否存在，最后将数据存储到指定的数据表中。其关键代码如下。

代码位置：光盘\TM\sl\22\bcty365\admin\savebccd.php

```
<?php include_once("../conn/conn.php");                //连接数据库
$bccdname=$_POST[bccdname];                            //获取 POST 方法提交的值
$owner=$_POST[owner];
$typeid=$_POST[typeid];
$content=$_POST[content];
$samepart=$_POST[samepart];
$addtime=date("Y-m-j H:i:s");                          //获取当前时间
    if(is_dir("./bccdimages")==false){                 //判断指定的文件是否存在
      mkdir("./bccdimages");                           //如果不存在，则创建一个新的文件夹
}
$link=date("YmjHis");                                  //获取当前时间
//为表单中提交的数据重新命名，以当前时间和随机数作为名称，其中使用$_FILES 获取表单中真实的名称，使用
//strstr 函数获取文件的后缀
    $path=$link.mt_rand(1000000,9999999).strstr($_FILES["imageaddress"]["name"],".");
$address="./bccdimages/".$path;                        //定义文件上传的路径
move_uploaded_file($_FILES["imageaddress"]["tmp_name"],$address);          //将文件上传到指定的文件中
$imageaddress="./admin/bccdimages/".$path;             //获取上传文件在服务器中的存储路径
//将表单中提交的数据存储到数据库中
$query=mysql_query("insert into tb_bccd(bccdname,owner,typeid,content,samepart,imageaddress,addtime)
values('$bccdname','$owner','$typeid','$content','$samepart','$imageaddress','$addtime')",$conn);
if($query==true){
      echo "<script>alert('编程词典添加成功!');history.back();</script>";
}else{
      echo "<script>alert('编程词典添加失败!');history.back();</script>";
}
?>
```

22.13.4　编辑编程词典的实现过程

在完成对编程词典信息的添加后，接下来就可以对编程词典的版本信息进行编辑，主要添加版本信息、价格、简介、功能和推出的服务。该模块的运行结果如图 22.48 所示。



图 22.48　编辑编程词典模块的运行结果

该功能的实现同样通过两个文件，一个是提交表单的文件 editbccd.php，另一个是处理表单提交数据的文件 sacvbccdbb.php。提交表单文件 editbccd.php 中使用的表单元素如表 22.6 所示。

表 22.6　编辑编程词典页中使用的重要表单元素

名　称	元素类型	重要属性	含　义
form1	form	method="post" action="savebccdbb.php" onSubmit="return chkinput(this)">	编辑编程词典表单
bccdname	text	class="txt_grey" disabled="disabled" value=" <?php $sql4=mysql_query("select bccdname from tb_bccd where id='".$_GET[bccdid]."'",$conn); $info4=mysql_fetch_array($sql4); echo unhtml($info4[bccdname]);　?>	编程词典名称，这里设置了文本的只读属性
bccdid	hidden	value="<?php echo $_GET[bccdid];?>	编程词典的 ID
bbid	select	<?php　$sql3=mysql_query("select * from tb_bb order by createtime desc ",$conn); $info3=mysql_fetch_array($sql3); if($info3==false){ echo　"<option>暂无版本信息</option>"; }else{ do{　?> <option　value="<?php echo $info3[id];?>"><?php echo unhtml($info3[bbname]);?></option> <?php }while($info3=mysql_fetch_array($sql3)); }?>	选择编程词典的版本
Submit	submit	value="添加" class="btn_grey"	提交表单

sacvbccdbb.php 文件对表单提交的数据进行处理，首先获取表单中提交的数据，然后判断指定的版本是否已经被添加，最后将数据存储到指定的数据表中。其代码如下。

代码位置：光盘\TM\sl\22\bcty365\admin\savebccdbb.php

```
<?php
$bccdid=$_POST[bccdid];                                         //获取表单中提交的数据
$bbid=$_POST[bbid];                                             //获取编程词典 id
$price=$_POST[price];                                           //获取编程词典单价
$content=$_POST[content];                                       //获取编程词典内容
$gn=$_POST[gn];                                                 //获取编程词典功能
$fw=$_POST[fw];                                                 //获取编程词典服务
include_once("../conn/conn.php");                               //连接数据库文件
//判断提交的编程词典是否已经被添加
$sql=mysql_query("select id from tb_bbqb where bccdid='".$bccdid."'",$conn);
$info=mysql_fetch_array($sql);                                  //检索指定编程词典的 id
if($info!=false){                                               //如果检索值为值，则弹出提示
     echo "<script>alert('该版编程词典已经添加!');history.back();</script>";
     exit; }
/* ****************************************************************** */
$query=mysql_query("insert into tb_bbqb(bccdid,bbid,price,content,gn,fw)
values('$bccdid','$bbid','$price','$content','$gn','$fw')",$conn);          //将表单中提交的数据存储到数据库中
//更新编程词典的价格
$querys=mysql_query("update tb_bccd set bbid='$bbid',price='$price' where id='".$bccdid."'");
if($query==true and $querys==true){                             //如果添加和更新操作为真，则弹出提示
     echo "<script>alert('版本信息添加成功!');history.back();</script>";
}else{                                                          //如果添加和更新操作为假，则弹出提示
     echo "<script>alert('版本信息添加失败!');history.back();</script>";
}
?>
```

22.14 软件升级管理模块设计

22.14.1 软件升级管理模块概述

软件升级管理模块实现对软件升级包的管理，其具体的功能包括添加升级包、编辑升级包、添加序列号和编辑序列号。软件升级管理模块中的添加升级包和添加序列号是一一对应的，其中根据所属的类别和版本来确定升级包对应的序列号，每一个版本一个类别的升级包对应一个序列号。

22.14.2 软件升级管理模块技术分析

在软件升级包管理模块中，应用到一个动态输出下拉列表框中值的技术。下面就来讲解一下该技术是如何实现的，在讲解该技术之前，先来了解下拉列表框的基本结构。



```
<select name="select"><!-->name 指定该下拉列表框的名称<!-->
    <!-->selected 设置下拉列表框的默认值,默认值为 PHP<!-->
    <option selected="selected">PHP</option>
    <!-->value 指定的"mysql"是下拉列表框传递的值,"MYSQL"为显示的内容<!-->
    <option value="mysql">MYSQL</option>
</select>
```

所谓动态输出下拉列表框中的值就是从数据库中读取数据，将获取到的数据输出到下拉列表框中，而不是直接在下拉列表框中设置某个固定的值。这里以软件升级管理模块 addsjb.php 文件中的“所属类别”下拉列表框为例进行讲解，其中设置下拉列表框的名称为 typeid，默认值为“请选择”，value 的值是从数据库中获取的 ID 值，显示的内容为从数据库中获取的类型名称。动态输出下拉列表框中值使用的关键代码如下。

代码位置：光盘\TM\sl\22\bcty365\ admin\wzdh.php

```
<!-- ------------------------------------设置下拉列表框的名称为 typeid------------------------------------ -->
<select name="typeid" class="txt_grey">
    <option value="" selected="selected">请选择</option>
    <!-- ----------------------------------------设置下拉列表框的名称为 typeid--------------------------- -->
    <?php
        include_once("../conn/conn.php");                          //连接数据库
        //从数据库中读取编程词典类型的数据
        $sql=mysql_query("select * from tb_type order by createtime desc",$conn);
        $info=mysql_fetch_array($sql);
        if($info==false){
            echo "<option >暂无类别</option>";
        }else{
            do{                                          //应用 do…while 循环语句输出类型的 ID 和类型的名称
                echo "<option value=".$info[id].">".$info[typename]."</option>";
            }
            while($info=mysql_fetch_array($sql));            //do…while 循环语句结束
        }
    ?>
</select>
```

下拉列表框不但可以动态输出数据库中某个字段的数据，而且可以输出数组中的数据。下面就实现一个在下拉列表框中动态输出数组中数据的功能，首先创建一个下拉列表框，然后设置下拉列表框的值，从数组中读取数据，应用 for 循环语句进行输出，其代码如下。

```
<!-- --------------------------------设置下拉列表框的名称为 select------------------------------------ -->
<select name="select" size="1">
<?php
    $string="ASP,PHP,JSP,.NET,DEL,VB,VC";                     //定义一个字符串
    $srtings=split(",",$string);                              //对字符串进行分割
    $count=count($srtings);                                   //获取数组中元素的数量
    for($i=0;$i<$count;$i++){                                 //根据数组中的元素的数量进行循环输出
        $result=$srtings[$i];                                 //定义变量，获取数组中指定的元素
    echo "<option>$result</option>";}                         //将数组中的元素输出到下拉列表框中
```

```
?>
</select>
```

动态输出数据库中数据到下拉列表框的运行结果如图 22.49 所示。



图 22.49　动态输出数据库中数据到下拉列表框

22.14.3　软件升级包上传的实现过程

软件升级包上传在添加升级包模块中实现，通过一个文件域文本框将升级包提交到服务器中指定的文件下，并且将该文件在服务器中的路径存储到数据库中，便于在前台实现对软件升级包的下载。其运行结果如图 22.50 所示。



图 22.50　软件升级包上传的运行结果

在本模块中通过 addsjb.php 文件来提交升级包的信息，通过 savesj.php 文件来对表单提交的数据进行处理。其中在将升级包上传到服务器的指定文件夹的过程中，主要应用的是 move_uploaded_file()函数。在 savesj.php 文件中，首先获取表单提交的数据，然后判断服务器中是否存在指定的文件，最后应用 move_uploaded_file()函数将升级包上传到指定的文件夹下，并且将数据存储到指定的数据表中。其程序代码如下。

代码位置：光盘\TM\sl\22\bcty365\admin\savesj.php

```
<?php
$name=$_POST[name];                         //获取表单提交的数据
$typeid=$_POST[typeid];                     //获取表单提交的数据
$content=$_POST[content];                   //获取表单提交的数据
```

```
$addtime=date("Y-m-j H:i:s");                                   //定义时间变量
$bbid=$_POST[bbid];                                             //获取表单提交的数据
if(is_dir("./sjxz")==false){                                    //判断指定的文件夹是否存在
      mkdir("./sjxz");                                          //如果指定的文件夹不存在，则创建一个指定的文件夹
}
$link=date("YmjHis");                                           //获取一个时间
$path=$link.mt_rand(1000000,9999999).strstr($_FILES["address"]["name"],".");      //重新设置升级包名称
$address="./sjxz/".$path;                                       //设置升级包在服务器中存储的指定路径
move_uploaded_file($_FILES["address"]["tmp_name"],$address);        //将升级包上传到指定的路径下
$address="./admin/sjxz/".$path;                                 //获取升级包在服务器中的存储路径
include_once("../conn/conn.php");                               //连接数据库文件
//将上传的数据存储到数据库中，这里将升级包在服务器中的路径存储到数据库中
$query=mysql_query("insert into tb_sjxz(name,typeid,content,addtime,address,bbid)
values('$name','$typeid','$content','$addtime','$address','$bbid')",$conn);
if($query){                                                     //如果添加操作成功，则弹出提示
      echo "<script>alert('升级包添加成功!');history.back();</script>";
}else{                                                               //如果添加操作失败，则弹出提示
      echo "<script>alert('升级包添加失败!');history.back();</script>";
}
?>
```

22.14.4　软件升级包删除的实现过程

软件升级包删除的实现主要根据当前数据中提供的 ID，执行 delete 删除语句，将数据表中相同 ID 的数据删除。其运行结果如图 22.51 所示。



图 22.51　软件升级包删除的运行结果

该功能的实现主要通过 editsjb.php 文件和 deletesjb.php 文件。通过 editsjb.php 文件输出数据库中存储的有关升级包的信息，以分页的形式显示，在每条记录的最后设置一个删除链接，通过脚本来调用 deletesjb.php 文件，根据变量中的 ID 值执行删除升级包的操作。其关键代码如下。

代码位置：光盘\TM\sl\22\bcty365\admin\deletesjb.php

```
<?php
$id=$_GET[id];                                                         //获取变量传递的 ID
include_once("../conn/conn.php");                                      //连接数据库
//执行删除操作，将数据表中对应的 ID 的数据删除
if(mysql_query("delete from tb_sjxz where id='".$id."'",$conn)){
    echo "<script>alert('该升级包删除成功!');history.back();</script>";    //如果删除操作成功，则弹出提示
}else{                                                                 //如果删除操作失败，则弹出提示
    echo "<script>alert('该升级包删除失败!');history.back();</script>";
}
?>
```

22.15　在 Linux 系统下发布网站

Linux 下发布基于 PHP 的网站，首先需要配置 PHP 的运行环境，其次需要对网卡参数进行设定。这里将以发布“BCTY365 网上社区”网站为例讲解 Linux 下如何实现网站的发布。假设已经申请到如表 22.7 所示的网络参数。

表 22.7　申请到的网络参数

参　　数	值
IP	192.168.1.*
Netmask	255.255.255.*
Network	192.168.1.0
Broadcast	192.168.1.*
Gateway	192.168.1.*
主机名	Tsoft
DNS	168.95.1*

Linux 下网站发布的操作步骤如下。

（1）配置 PHP 的运行环境，在 22.4 节中已经做了详细介绍，这里不再赘述。

（2）将“BCTY365 网上社区”网站的所有文件复制到 Apache 主目录下。

（3）设置主机名称。在终端窗口中输入如下命令编辑/etc/sysconfig/network 文件。

```
vi /etc/sysconfig/network
```

将该文件中的参数 NETWORKING 设置为 yes，表示启动网络，将参数 HOSTNAME 设置为 Tsoft，表示设置主机名为 Tsoft。

（4）设置网卡参数。在终端窗口中输入如下命令编辑文件/etc/sysconfig/network-scripts/ifcfg-eth0。

```
vi   /etc/sysconfig/network-scripts/ifcfg-eth0
```

该文件的相关参数设置如表 22.8 所示。



表 22.8　设置网卡的相关参数

参　　数	说　　明
DEVICE=eth0	设置网卡名称，要与 ifcfg-eth0 对应
ONBOOT=yes	指定在开机时启动网卡
BOOTPROTO=static	设定启动时获取 IP 的方式
IPADDR=192.168.1.*	设定服务器 IP 地址
NETMASK=255.255.255.*	设定子网掩码
BROADCAST=192.168.1.*	设定同网段的广播地址
GETWAY=192.168.1.*	设定网卡的网关

（5）设置 DNS 主机的 IP。在终端编辑/etc/resolv.conf 文件：

```
vi /etc/resolv.conf
```

设置参数 nameserver 的值为 168.95.1.*。

（6）重新启动网络设置。在终端窗口中输入如下命令。

```
/etc/rc.d/inin.d/network restart
ifdow eth0
ifup eth0
```

（7）打开浏览器，在地址栏中输入服务器 IP 地址或域名，打开如图 22.52 所示的页面，说明 Linux 下“BCTY365 网上社区”网站发布成功。



图 22.52　Linux 下“BCTY365 网上社区”网站运行结果

22.16 开发技巧与难点分析

22.16.1 管理员权限的设置

为了更好地管理和维护网站的论坛，针对论坛设置了一个管理员，该管理员不在后台进行操作，而是在前台为管理员设置特殊的权限，也可以称之为版主。其实现的原理是：首先在数据库中设置不同的值代表不同的权限，“0”代表普通会员，“1”代表管理员；然后在论坛的页面中进行判断，当用户的类型为“1”时，不但具有普通会员的权限，而且具有删除发布帖子、回复帖子和顶帖的权限；如果用户的类型不是“1”，则不具有上述的权限，只能是发布和回复帖子。管理员和普通会员登录的页面效果是不同的，如图 22.53 和图 22.54 所示。



图 22.53 管理员登录的操作页面



图 22.54 普通会员登录的操作页面

在页面中执行的判断语句判断登录用户的类型，然后根据类型判断用户的权限。其程序关键代码如下。

```
<?php
    if($_SESSION["unc"]!=""){                                //判断 session 变量的值是否为空
        //根据 session 变量的值获取该用户的类型
        $sqlu=mysql_query("select usertype from tb_user where usernc='".$_SESSION["unc"]."'",$conn);
```

```
            $infou=mysql_fetch_array($sqlu);
            if($infou["usertype"]==1){                    //判断当该用户的类型等于 1 时执行下面的内容
?>
<!-->调用 JavaScript 脚本，执行删除发布帖子的操作<!-->
<img src="images/lt_15(10).jpg" onclick="javascript:if(window.confirm('您确定删除该帖么？
')==true){window.location.href='bbs_delete.php?id=<?php echo $infob["id"]?>';}" style="cursor:hand"/>
<?php
        }
            //如果用户的类型不是 1，则不执行上述的内容
    }
?>
```

22.16.2　帖子置顶的设置

所谓帖子置顶就是将某个指定的帖子在对应的版块中最前面的位置显示，该权限只有管理员才拥有，普通会员不具备该权限。其实现的原理如下。

首先，在数据库中存储发布帖子信息的数据表中设置一个字段 top，指定该字段为数字类型，其默认值为 0。

然后，在网页中判断登录用户的权限，如果是管理员，则具有帖子置顶的权限，否则将弹出提示对话框“对不起，您不具备该操作权限！”。

最后，如果是管理员，则执行 settop.php 文件，根据对应帖子的 ID 查找到发布帖子信息表中对应的数据，更新该条数据中 top 字段的值为 1。

判断登录用户权限使用的代码如下。

```
<?php
    if($_SESSION["unc"]==""){                 //判断 session 变量的值是否为空，如果为空则执行下面的脚本
        echo "javascript:alert('请先登录本站，然后进行此操作!');window.location.href='index.php';";
    }else{
        //如果不为空则执行下面的内容，判断登录用户的权限
        $sqlu=mysql_query("select usertype from tb_user where usernc='".$_SESSION["unc"]."'",$conn);
        $infou=mysql_fetch_array($sqlu);
        if($infou["usertype"]==1){            //如果登录用户的类型是"1"，则说明是管理员，则执行下面的内容
            echo "javascript:window.location.href='settop.php?id=".$infob["id"]."'";
        }else{
            //如果登录用户不是管理员，则执行下面的内容
            echo "javascript:alert('对不起，您不具备该操作权限!');";
        }
    }
?>
```

实现帖子置顶是通过 settop.php 文件来完成的，在该文件中，根据变量提交的值获取到发布帖子信息表中对应的数据，更新该条数据中字段 top 的值，并且对该字段的值进行判断。如果字段 top 的值为 1，则说明该帖已经置顶，此时将字段的值更新为 0，即取消置顶；如果字段 top 的值为 0，则说明该帖没有进行置顶，此时将字段的值更新为 1，即置顶该帖。settop.php 文件的程序代码如下。

```
<?php    include_once("conn/conn.php");                                    //连接数据库文件
//根据获取的 ID 值，从数据表中读取到对应的数据
$sql=mysql_query("select top from tb_bbs where id='".$_GET["id"]."'",$conn);
$info=mysql_fetch_array($sql);
if($info[top]==1){          //判断对应数据记录中的字段 top 的值，如果字段 top 的值为"1"，则执行下面的内容

    mysql_query("update tb_bbs set top=0 where id='".$_GET["id"]."'",$conn);      //更新字段 top 的值为 0
}elseif($info[top]==0){              //如果对应数据记录中的字段 top 的值为"0"，则执行下面的内容
    mysql_query("update tb_bbs set top=1 where id='".$_GET["id"]."'",$conn);      //更新字段 top 的值为 1
}
echo "<script>alert('置上设置成功!');history.back();</script>";
?>
```

22.16.3　解决提示“客户反馈发表失败！”的问题

在对客户反馈模块进行测试的过程中，当提交反馈信息时，提示“客户反馈发表失败！”。经过初步的判断，问题可能出现在向数据库添加数据的过程中。为进一步查找出问题的根源，在 saveleaveword.php 页中应用了 mysql_error()函数，该函数可获取详细的错误信息，将它放置在 insert 添加语句之后，然后重新运行网页，运行结果如图 22.55 所示。



图 22.55　应用 mysql_error()函数获取的错误信息

在该结果图中可以看出是数据表中的 type 字段出了问题，经过仔细检查发现，原来是在编写 insert 添加语句的过程中，将语句中字段的排列顺序与对应的字段值的顺序弄错了，如图 22.56 所示。

```
//向数据库中添加数据
if(mysql_query("insert into tb_leaveword(userid,type,title,content,createtime)
                    values('$userid','$title','$type','$content','".date("Y-m-j H:i:s")."')",$conn)){
    echo "<script>alert('留言发表成功！');history.back();</script>";
}else{
    echo "<script>alert('留言发表失败！');history.back();</script>";
}
```

颠倒$title 和$type 的顺序

图 22.56　代码中的错误截图

调整变量$title 和$type 的顺序，重新运行程序，提示“留言添加成功”。

22.16.4　解决指定商品没有被删除的问题

在在线订购模块的测试过程中，当执行删除购物车中某个商品的操作时，发现该项操作没能正确



执行，指定的商品没有被删除。分析出错原因：首先，从该功能实现的思路入手，并且仔细核对其中使用的变量以及函数是否正确。

删除商品的操作是根据商品的 ID 来进行处理的，当单击“删除”超链接时，将指定商品的 ID 通过变量$ides 传入到执行删除商品操作的文件 delgoods.php 中，然后在 delgoods.php 文件中，获取变量$ides 传入的 ID 值，并且根据 ID 值完成删除购物车中指定商品的操作。在查看该功能实现的过程中发现，定义的变量$ides 与 delgoods.php 文件中$_GET["id"]获取的变量值不一致，从而导致删除商品的操作没有正确执行，代码的出错位置如图 22.57 所示。



图 22.57　代码中的错误显示

查找出错误原因，将定义的变量与获取的变量值统一使用 id，保存文件，重新运行程序，删除操作可以正常运行。

22.16.5　解决发帖和回帖时上传的图片不能够正常显示的问题

在测试网上社区的论坛模块时，发现发帖和回帖时上传的图片不能够正常显示。运行结果如图 22.58 所示。分析错误原因：主要有两个方面，一是图片没有上传到指定的服务器文件夹下；二是上传成功后，没能正确地读取数据库中指定图片的路径。



图 22.58　程序运行错误结果

首先，检测第一种情况，没有出现问题，图片可以上传到指定的文件夹下，并且图片的路径也可以存储到指定的数据表中，从而排除了第一种情况的可能。

然后，看第二种情况，查看获取的图片路径是否正确。发现在读取数据库中图片路径的代码段中，使用了错误的字段名称，数据库中图片路径存储使用的字段名是 photo，而在程序代码段中使用的却是 photos。错误代码如下。

```
<?php
    if($infob[photo]!=""){                                    //判断是否存在图片
        $photos=substr($infob[photos],2,70);                  //获取图片存储的路径
        echo (stripslashes($infob["content"]));               //输出图片的内容
        echo "<img src=\"$photos\">";                         //输出图片
    }else{
        echo (stripslashes($infob["content"]));               //输出帖子内容
    }
?>
```

将代码段中的字段名进行修改，然后重新运行程序，图片正常显示。

22.17　小　　结

本章从项目开发的实际角度出发，以某科技公司的实际需求为背景，详细地讲解了“BCTY365 网上社区”系统的开发过程，其中以系统的整体开发流程为主线，重点介绍技术支持、在线订购、社区论坛和编程词典等几个大模块的实现方法，并且对管理员权限设置、帖子置顶设置进行讲解，而且在本章中还讲解了在 Linux 下如何搭建 PHP 的开发环境以及在 Linux 下如何发布网站。

第23章 Struts 2+Spring+Hibernate+MySQL实现网络商城

喜欢网上购物的读者一定登录过淘宝，也一定被网页上琳琅满目的商品所吸引，忍不住拍一个自己喜爱的商品。如今也有越来越多的人加入到网购的行列，做网上店铺的老板，做新时代的购物潮人，你是否也想过开发一个自己的网上商城？本章我们将一起进入Go购网络商城开发的旅程。

通过阅读本章，读者可以：

- 了解网上商城的核心业务
- 掌握网站开发的基本流程

23.1　开 发 背 景

近年来，随着 Internet 的迅速崛起，互联网用户的爆炸式增长以及互联网对传统行业的冲击让其成为人们快速获取、发布和传递信息的重要渠道。于是电子商务逐渐流行起来，越来越多的商家在网上建起网上商城，向消费者展示出一种全新的购物理念，同时也有越来越多的网友加入到了网上购物的行列。阿里巴巴旗下的淘宝的成功展现了电子商务网站强大的生命力和电子商务网站更加光明的未来。

为了利用 Internet 这个平台，实现一种全新的购物方式——网上购物，其目的是方便广大网友购物，让网友足不出户就可以逛商城买商品，为此构建 Go 购商城系统。

23.2　系 统 分 析

23.2.1　需求分析

Go 购商城系统是基于 B/S 模式的电子商务网站，用于满足不同人群的购物需求，笔者通过对现有的商务网站的考察和研究，从经营者和消费者的角度出发，以高效管理、满足消费者需求为原则，要求本系统满足以下要求。

（1）统一友好的操作界面，具有良好的用户体验；

（2）商品分类详尽，可按不同类别查看商品信息；

（3）推荐产品、人气商品以及热销产品的展示；

（4）会员信息的注册及验证；

（5）用户可通过关键字搜索指定的产品信息；

（6）用户可通过购物车一次购买多件商品；

（7）实现收银台的功能，用户选择商品后可以在线提交订单；

（8）提供简单的安全模型，用户必须先登录，才允许购买商品；

（9）用户可查看自己的订单信息；

（10）设计网站后台，管理网站的各项基本数据；

（11）系统运行安全稳定、响应及时。

23.2.2　可行性分析

在正式开发系统之前，首先需要对 Go 购商城的技术、操作和经济成本 3 个方面进行可行性



分析。

1. 技术可行性

本系统采用 MVC 设计模式，使用当前最流行的 Struts 2+Spring 2+Hibernate 框架进行开发，在前台用 JSP 进行页面开发和管理用户界面，利用轻巧的 JavaScript 库——jQuery 处理页面的 JavaScript 脚本，使开发更加高效，提示信息更加完善，界面友好，具有较强的亲和力。后台采用 MySQL 数据库，MySQL 小巧高效的特点可以满足系统的性能要求。本系统使用当前主流的 Java 开源开发工具 Eclipse 和 Tomcat 服务器进行程序开发和发布，它们是完全免费的，可以节约开发成本。本系统采用的技术和开发环境在实际开发中应用非常广泛，充分说明本系统在技术方面可行。

2. 操作可行性

Go 购商城主要面向的是喜欢网购的网友，只要 Go 购商城的用户会一些简单的计算机操作，就可以进行网上购物，不需要用户具有较高的计算机专业知识，而且对于网站基本信息的维护也是十分简单，管理员可以在任何一台可以上网的机器上对网站进行维护，网站的简单易用性充分说明了 Go 购商城的操作可行性。

3. 经济成本可行性

在实际的销售运营过程中，产品的宣传受到限制，采购商或顾客只能通过上门咨询、电话沟通等方式进行各种产品信息的获取，而且时间与物理的局限性严重影响了产品的销售，并且在无形中提高了产品的销售成本。本系统完全可以改变这种现状，以少量的时间和资金建立企业商务网络，以此来使企业与消费者之间的经济活动变得更加灵活、主动。

系统中应用的开发工具以及技术框架都是免费的，这无疑为网站的成本再一次压缩了空间，从成本可行性分析来看，该系统充分体现了将产品利益最大化的企业原则。

23.3　系 统 设 计

23.3.1　功能结构图

Go 购商城系统分为前台和后台两个部分的操作。前台主要有两大功能，分别是展示产品信息的各种浏览操作和会员用户购买商品的操作，当会员成功登录后，就可以使用购物车进行网上购物。Go 购商城前台功能结构图如图 23.1 所示。

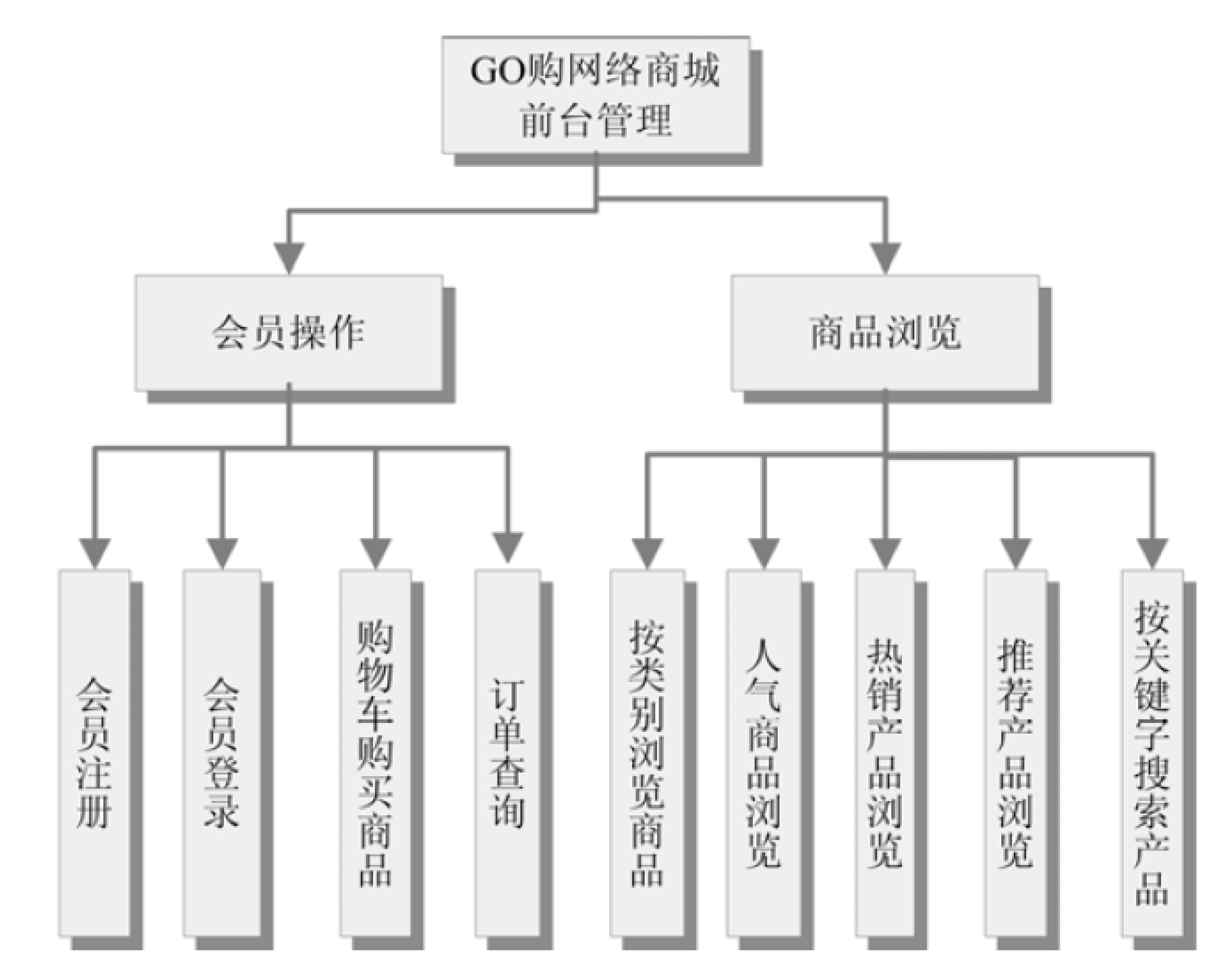


图 23.1　Go 购商城系统前台功能结构图

后台的主要功能是当管理员成功登录后台后，用户可以对网站的基本信息进行维护。例如，管理员可以对商品的类别进行管理，如可以删除和添加产品的类别，可以对商品信息进行维护，可以添加、删除、修改和查询产品信息，并上传产品的相关图片，可以对会员的订单进行集中管理，管理员可以对订单信息进行自定义的条件查询并修改指定的产品信息。Go 购商城后台功能结构图如图 23.2 所示。

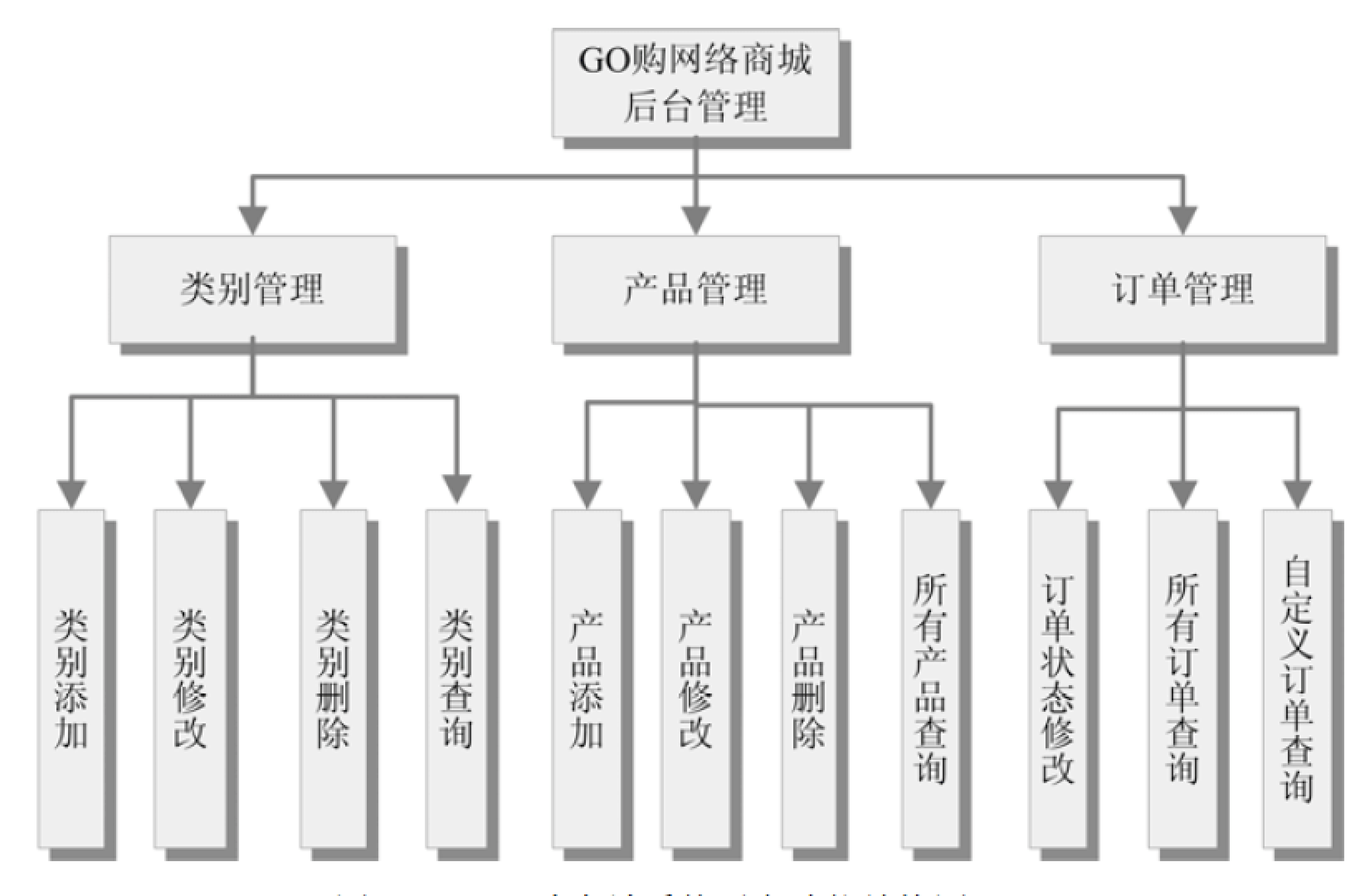


图 23.2　Go 购商城系统后台功能结构图

23.3.2　系统流程图

在 Go 购商城中只有会员才允许进行购物操作，所以初次登录网站的用户如果想进行购物操作必须注册为 Go 购商城的会员。成功注册为会员后，会员可以使用购物车选择自己需要的商品，在确认订单付款后，系统将自动生成此次交易的订单基本信息。网站基本信息的维护由网站管理员负责，由管理员负责对商品信息、商品类别信息以及订单信息进行维护，关于订单的维护只能修改订单的状态，并不能修改订单的基本信息，因为订单确认之后就是用户与商家之间交易的凭证，第三方无权修改。

Go 购商城的系统流程图如图 23.3 所示。

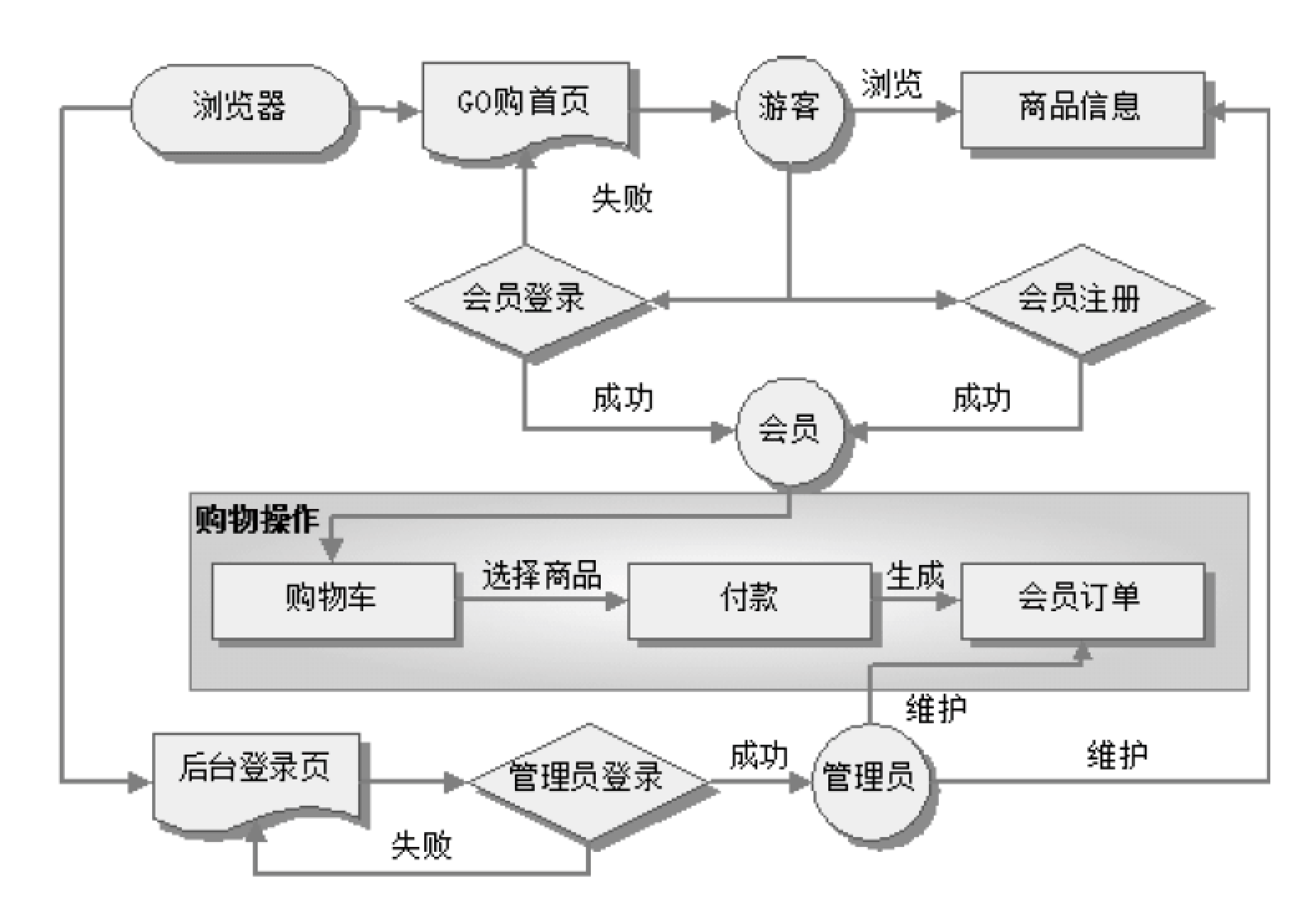


图 23.3　Go 购商城系统流程图

23.3.3　开发环境

在进行 Go 购商城网站开发时，需要具备以下开发环境。

1.服务器端

操作系统：Windows 2003 或者更高版本的服务器操作系统。

Web 服务器：Tomcat 7.0 或 7.0 以上版本。

Java 开发包：JDK 1.7 以上。

数据库：MySQL。

浏览器：IE 6.0 或者更高版本的浏览器。
分辨率：最低要求为 1024×768 像素。

2.客户端

浏览器：IE 6.0 或者更高版本的浏览器。
分辨率：最低要求为 1024×768 像素。

23.3.4　文件夹组织结构

在编写代码之前，可以把系统中可能用到的文件夹先创建出来（例如，创建一个名为 images 的文件夹，用于保存网站中所使用的图片），这样不但可以方便以后的开发工作，也可以规范网站的整体架构。本系统的文件夹组织结构如图 23.4 所示。



图 23.4　Go 购商城文件夹组织结构

23.3.5　系统预览

系统预览将以用户交易为例，列出几个关键的页面，商品交易是 Go 购商城的核心模块之一，通过该预览的展示，读者可以对 Go 购商城有个基本的了解，同时读者也可以在光盘中对本程序的源程序进行查看。

当用户在地址栏中输入 Go 购商城的域名，就可以进入 Go 购商城，首页将商品的类别信息分类展现给用户，并在首页展示部分的人气商品、推荐商品、热销商品以及上市新品，首页部分效果如图 23.5 所示。



图 23.5　Go 购商城首页部分页面效果图

如果用户为会员登录后就可以直接进行产品的选购，当用户在商品信息详细页面中单击“直接购买”超链接，就会将该商品放入购物车中，同时用户也可以使用购物车选购多种商品，购物车同时可以保存多件会员采购的商品信息。图 23.6 为用户选购多件产品的效果。



图 23.6　Go 购商城购物车页面效果图

当用户到收银台付款后，系统将自动生成订单，会员可通过单击左侧导航栏中的“我的订单”超

链接查看自己的订单信息，如图 23.7 所示。



图 23.7　Go 购商城会员订单信息效果图

23.4　数据库设计

整个应用系统的运行离不开数据库的支持，数据库可以说是应用系统的灵魂，没有了数据库的支撑，系统只能说是一个空架子，它将很难完成与用户之间的交互。由此可见，数据库在系统中占有十分重要的地位。本系统采用的是 MySQL 数据库，通过 Hibernate 实现系统的持久化操作。

本节将根据 Go 购商城网站的核心实体类，分别设计对应的 E-R 图和数据表。

23.4.1　数据库概念设计

所谓的数据库概念化设计，就是将现实世界中的对象以 E-R 图的形式展现出来，本节将对程序所应用到的核心实体对象设计对应的 E-R 图。

（1）会员信息表 tb_customer 的 E-R 图，如图 23.8 所示。

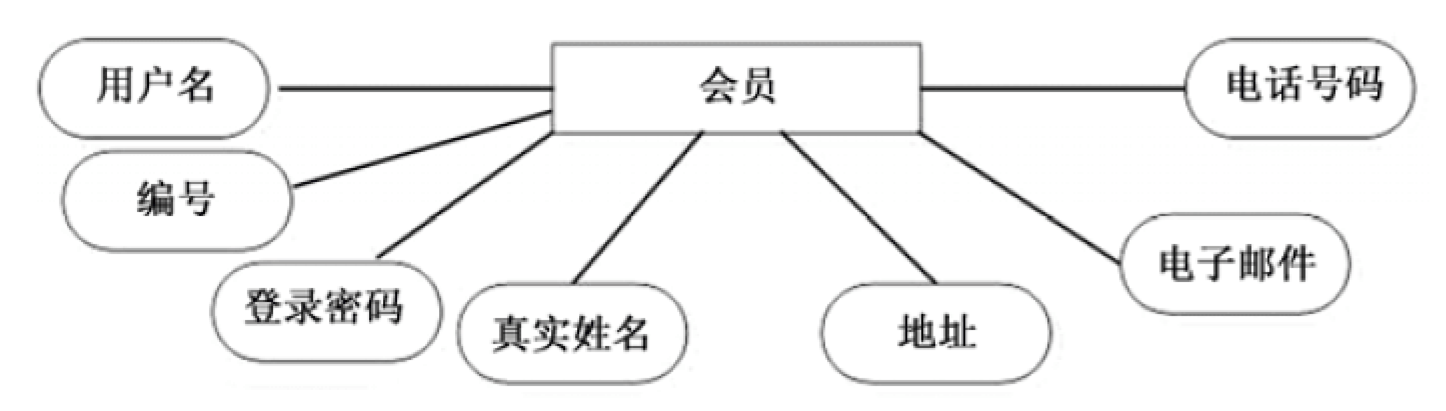


图 23.8　会员信息表 tb_customer 的 E-R 图

（2）订单信息表 tb_order 的 E-R 图，如图 23.9 所示。



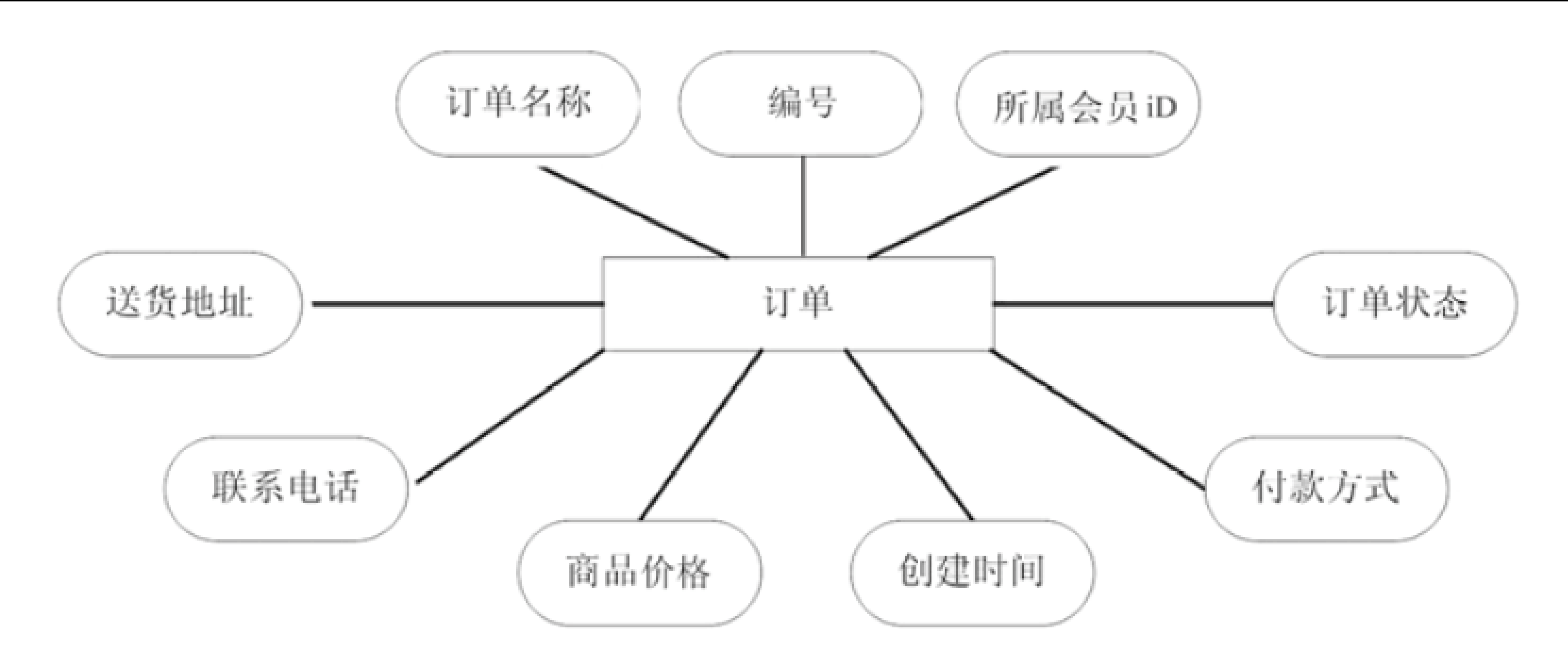


图 23.9　订单信息表 tb_order 的 E-R 图

（3）订单条目信息表 tb_orderitem 的 E-R 图，如图 23.10 所示。

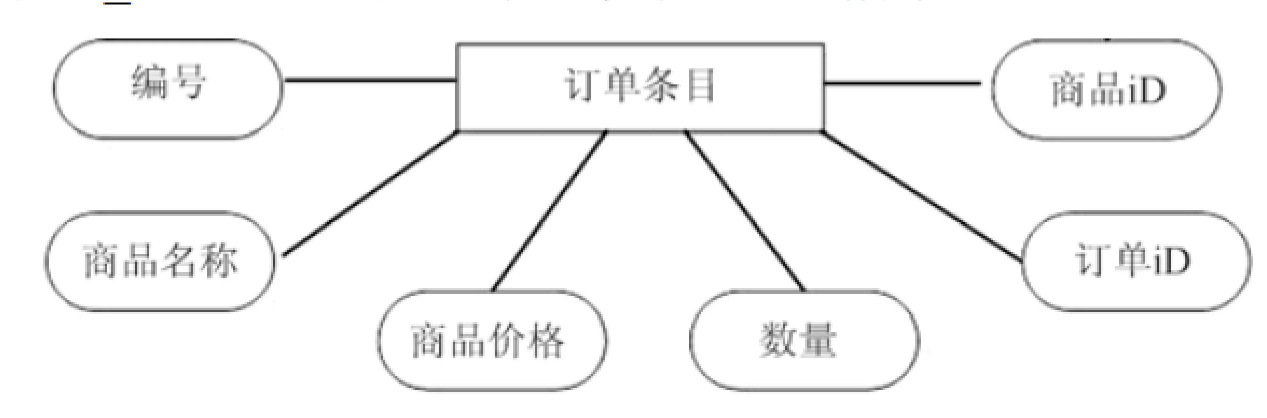


图 23.10　订单条目信息表 tb_orderitem 的 E-R 图

（4）商品信息表 tb_productinfo 的 E-R 图，如图 23.11 所示。

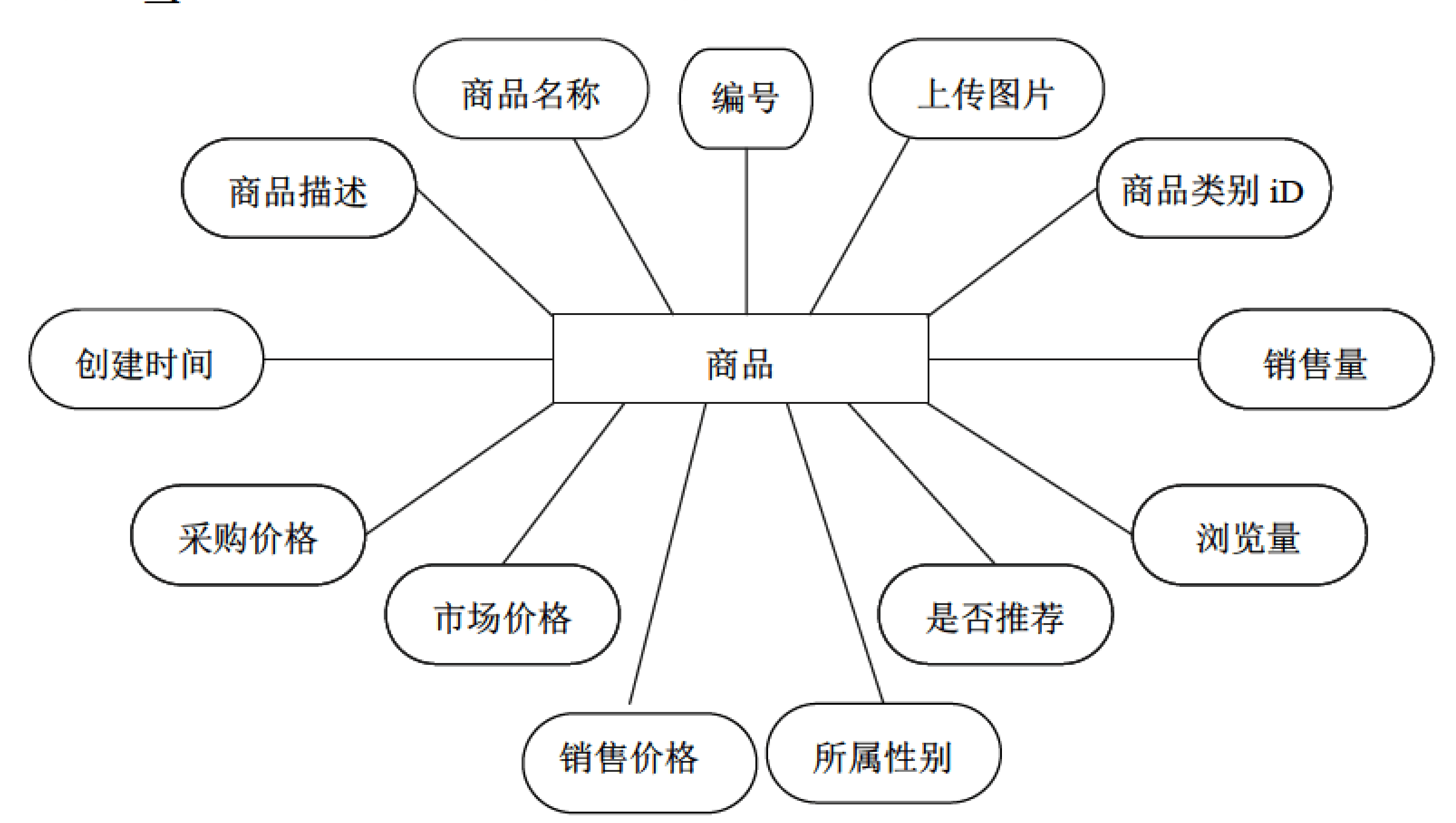


图 23.11　商品信息表 tb_productinfo 的 E-R 图

（5）商品类别信息表 tb_productcategory 的 E-R 图，如图 23.12 所示。

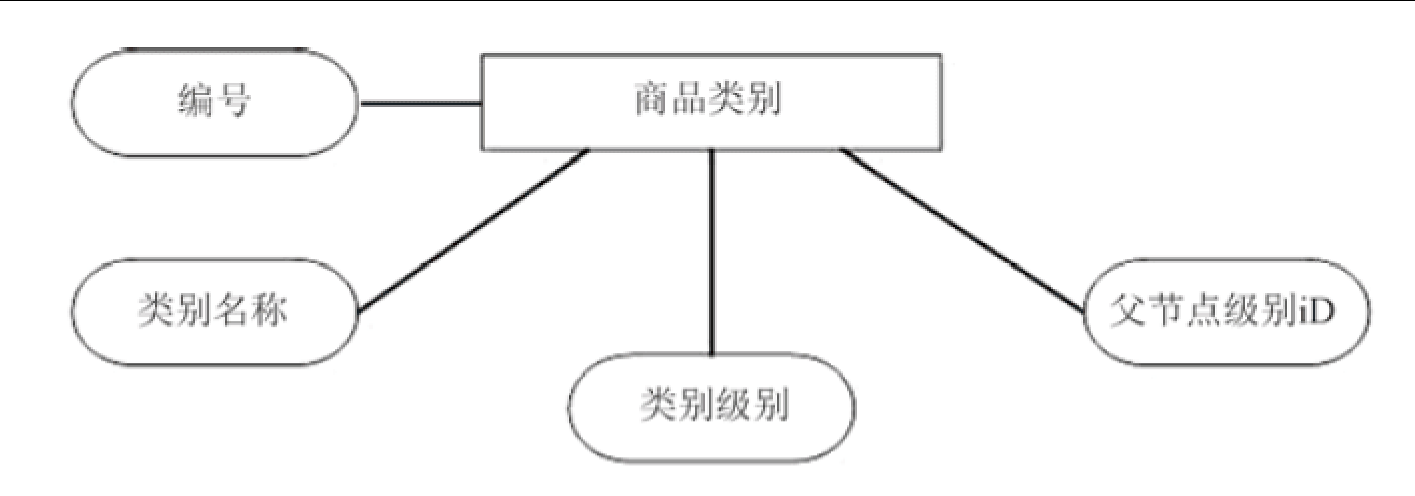


图 23.12　商品类别信息表 tb_productcategory 的 E-R 图

23.4.2　创建数据库及数据表

本系统采用 MySQL 数据库，创建的数据库名称为 db_database24，数据库 db_database24 中包含 7 张数据表。所有数据表的定义如下。

1. tb_customer（会员信息表）

用于存储会员的注册信息，该表的结构如表 23.1 所示。

表 23.1　tb_customer 信息表的表结构

字 段 名	数 据 类 型	是 否 为 空	是 否 主 键	默 认 值	说　明
id	INT(10)	否	是	NULL	系统自动编号
username	VARCHAR(50)	否	否	NULL	会员名
password	VARCHAR(50)	否	否	NULL	登录密码
realname	VARCHAR(20)	是	否	NULL	真实姓名
address	VARCHAR(200)	是	否	NULL	地址
email	VARCHAR(50)	是	否	NULL	电子邮件
mobile	VARCHAR(11)	是	否	NULL	电话号码

2. tb_order（订单信息表）

用于存储会员的订单信息，该表的结构如表 23.2 所示。

表 23.2　tb_order 信息表的表结构

字 段 名	数 据 类 型	是 否 为 空	是 否 主 键	默 认 值	说　明
id	INT(10)	否	是	NULL	系统自动编号
name	VARCHAR(50)	否	否	NULL	订单名称
address	VARCHAR(200)	否	否	NULL	送货地址
mobile	VARCHAR(11)	否	否	NULL	联系电话
totalPrice	FLOAT	是	否	NULL	商品价格
createTime	DATETIME	是	否	NULL	创建时间
paymentWay	VARCHAR(15)	是	否	NULL	付款方式
orderState	VARCHAR(10)	是	否	NULL	订单状态
customerId	INT(11)	是	否	NULL	会员 iD

3. tb_orderitem（订单条目信息表）

用于存储会员订单的条目信息，该表的结构如表 23.3 所示。

表 23.3　tb_orderitem 信息表的表结构

字 段 名	数 据 类 型	是 否 为 空	是 否 主 键	默 认 值	说 明
id	INT(10)	否	是	NULL	系统自动编号
productId	INT(11)	否	否	NULL	商品 iD
productName	VARCHAR(200)	否	否	NULL	商品名称
productPrice	FLOAT	否	否	NULL	商品价格
amount	INT(11)	是	否	NULL	商品数量
orderId	VARCHAR(30)	是	否	NULL	订单 iD

4. tb_productinfo（商品信息表）

用于存储商品信息，该表的结构如表 23.4 所示。

表 23.4　tb_productinfo 信息表的表结构

字 段 名	数 据 类 型	是 否 为 空	是 否 主 键	默 认 值	说 明
id	INT(10)	否	是	NULL	系统自动编号
name	VARCHAR(100)	否	否	NULL	商品名称
description	TEXT	是	否	NULL	商品描述
createTime	DATETIME	是	否	NULL	创建时间
baseprice	FLOAT	是	否	NULL	采购价格
marketprice	FLOAT	是	否	NULL	市场价格
sellprice	FLOAT	是	否	NULL	销售价格
sexrequest	VARCHAR(5)	是	否	NULL	所属性别
commend	BIT(1)	是	否	NULL	是否推荐
clickcount	INT(11)	是	否	NULL	浏览量
sellCount	INT(11)	是	否	NULL	销售量
categoryId	INT(11)	是	否	NULL	商品类别 iD
uploadFile	INT(11)	是	否	NULL	上传图片 iD

5. tb_productcategory（商品类别信息表）

用于存储商品的类别信息，该表的结构如表 23.5 所示。

表 23.5　tb_productcategory 信息表的表结构

字 段 名	数 据 类 型	是 否 为 空	是 否 主 键	默 认 值	说 明
id	INT(10)	否	是	NULL	系统自动编号
name	VARCHAR(50)	否	否	NULL	类别名称
level	INT(11)	是	否	NULL	类别级别
pid	INT(11)	是	否	NULL	父节点类别 iD

6. tb_user（管理员信息表）

用于存储网站后台管理员信息，该表的结构如表 23.6 所示。

表 23.6　tb_user 信息表的表结构

字　段　名	数 据 类 型	是 否 为 空	是 否 主 键	默　认　值	说　　明
id	INT(10)	否	是	NULL	系统自动编号
username	VARCHAR(50)	否	否	NULL	用户名
password	VARCHAR(50)	否	否	NULL	登录密码

7. tb_uploadfile（上传文件信息表）

用于存储上传文件的路径信息，该表的结构如表 23.7 所示。

表 23.7　tb_uploadfile 信息表的表结构

字　段　名	数 据 类 型	是 否 为 空	是 否 主 键	默　认　值	说　　明
id	INT(10)	否	是	NULL	系统自动编号
path	VARCHAR(255)	否	是	NULL	文件路径信息

23.5　公共模块的设计

在项目中经常会有一些公共类，例如 Hibernate 的初始化类，一些自定义的字符串处理方法，抽取系统中公共模块更加有利于代码重用，同时也能提高程序的开发效率。在进行正式开发时首先要进行公共模块的编写。下面介绍 Go 购商城的公共类。

23.5.1　泛型工具类

Hibernate 提供了高效的对象到关系型数据库的持久化服务，通过面向对象的思想进行数据持久化的操作，Hibernate 的操作对象就是数据表所对应的实体对象，为了将一些公用的持久化方法提取出来，首先需要实现获取实体对象的类型方法，在本应用中通过自定义创建一个泛型工具类 GenericsUtils 来达到此目的，其代码如下。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\util\HibernateUtils

```
public class GenericsUtils {
	@SuppressWarnings("unchecked")
	public static Class getGenericType(Class clazz){
	Type genType = clazz.getGenericSuperclass();		//得到泛型父类
	Type[] types = ((ParameterizedType) genType).getActualTypeArguments();
	if (!(types[0] instanceof Class)) {
			return Object.class;
		}
	return (Class) types[0];
```



```
    }
    //获取对象的类名称
    @SuppressWarnings("unchecked")
    public static String getGenericName(Class clazz){
    return clazz.getSimpleName();
    }
}
```

23.5.2　数据持久化类

在本应用中利用 DAO 模式实现数据库基本操作方法的封装，数据库中最为基本的操作就是增、删、改、查，据此自定义数据库操作的公共方法。由控制器负责获取请求参数并控制转发，由 DAOSupport 类组织 SQL 语句。

根据自定义的数据库操作的公共方法，创建接口 BaseDao<T>，关键代码如下。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\dao\BaseDao

```
public interface BaseDao<T> {
    //基本数据库操作方法
    public void save(Object obj);                                           //保存数据
    public void saveOrUpdate(Object obj);                                   //保存或修改数据
    public void update(Object obj);                                         //修改数据
    public void delete(Serializable ... ids);                               //删除数据
    public T get(Serializable entityId);                                    //加载实体对象
    public T load(Serializable entityId);                                   //加载实体对象
    public Object uniqueResult(String hql, Object[] queryParams);           //使用 hql 语句操作
}
```

创建类 DaoSupport，该类继承 BaseDao<T>接口，在类中实现接口中自定义的方法，其关键代码如下。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\dao\DaoSupport

```
public abstract class DaoSupport<T> implements BaseDao<T>{
    protected Class<T> entityClass = GenericsUtils.getGenericType(this.getClass());     //泛型的类型
    @Autowired
    protected HibernateTemplate template;                                               // Hibernate 模板
    public HibernateTemplate getTemplate() {
    return template;
    }
    //删除指定的对象信息
    public void delete(Serializable ... ids) {
    for (Serializable id : ids) {                               //遍历标识参数
    T t = (T) getTemplate().load(this.entityClass, id);         //加载指定对象
    getTemplate().delete(t);                                    //执行删除操作
    }
    }
    //利用 get()方法加载对象，获取对象的详细信息
    @Transactional(propagation=Propagation.NOT_SUPPORTED,readOnly=true)
```

```
        public T get(Serializable entityId) {
        return (T) getTemplate().get(this.entityClass, entityId);          //加载指定对象
        }
        //利用 load()方法加载对象，获取对象的详细信息
        @Transactional(propagation=Propagation.NOT_SUPPORTED,readOnly=true)
        public T load(Serializable entityId) {
        return (T) getTemplate().load(this.entityClass, entityId);         //加载指定对象
        }
        //利用 save()方法保存对象的详细信息
        public void save(Object obj) {
        getTemplate().save(obj); //
        }
        //保存或修改信息
        public void saveOrUpdate(Object obj) {
        getTemplate().saveOrUpdate(obj);
        }
        //利用 update()方法修改对象的详细信息
        public void update(Object obj) {
        getTemplate().update(obj);
        }
}
```

23.5.3　分页操作

分页查询是 Java Web 开发中十分常用的技术。在数据库量非常大的情况下，不适合将所有数据显示到一个页面之中，这样既给查看带来不便，又占用程序及数据库的资源，此时就需要对数据进行分页查询。本系统应用 Hibernate 的 find 方法实现数据分页的操作，将分页的方法封装在创建类 DaoSupport 中，下面将介绍 Hibernate 分页实现的方法。

1. 分页实体对象

首先定义分页的实体对象，封装分页基本属性信息和在分页过程中使用的获取页码的方法。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\model\PageModel<T>

```
public class PageModel<T> {
        private int totalRecords;                                          //总记录数
        private List<T> list;                                              //结果集
        private int pageNo;                                                //当前页
        private int pageSize;                                              //每页显示多少条
        //取得第一页
        public int getTopPageNo() {
        return 1;
        }
        //取得上一页
        public int getPreviousPageNo() {
        if (pageNo <= 1) {
        return 1;
```

```
        }
        return pageNo -1;
        }
        //取得下一页
        public int getNextPageNo() {
        if (pageNo >= getTotalPages()) {                    //如果当前页大于页码
        return getTotalPages() == 0 ? 1 : getTotalPages();  //返回最后一页
        }
        return pageNo + 1;
        }
         //取得最后一页
        public int getBottomPageNo() {
        return getTotalPages() == 0 ? 1 : getTotalPages();  //如果总页数为 0 返回 1，反之返回总页数
        }
        //取得总页数
        public int getTotalPages() {
        return (totalRecords + pageSize - 1) / pageSize;
        }
        …                                                   //省略的 Setter 和 Getter 方法
}
```

在页面的实体对象中，封装了几个重要的页码获取方法，即获取第一页、上一页、下一页、最后一页以及总页数的方法。

在取得上一页页码的方法 getPreviousPageNo()中，如果当前页的页码数为首页，那么上一页返回的页码数为 1。

在获取最后一页的方法 getBottomPageNo()中，通过三目运算符进行选择判断返回的页码，如果总页数为 0 则返回 1，反之返回总页面数。当数据库中没有任何信息的时候，总页数为 0。

2. 实现自定义分页方法

在公共接口中定义几种不同的分页方法，这些方法定义使用了相同的分页方法，不同的参数，自定义分页方法关键代码如下。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\dao\BaseDao

```
public interface BaseDao<T> {
        …                                                   //基本数据库操作方法
        //分页操作
        public long getCount();                             //获取总信息数
        public PageModel<T> find(int pageNo, int maxResult);  //普通分页操作
        //搜索信息分页方法
        public PageModel<T> find(int pageNo, int maxResult,String where, Object[] queryParams);
        //按指定条件排序分页方法
        public PageModel<T> find(int pageNo, int maxResult,Map<String, String> orderby);
        //按指定条件分页和排序的分页方法
        public PageModel<T> find(String where, Object[] queryParams,
                Map<String, String> orderby, int pageNo, int maxResult);
}
```

23.5.4　字符串工具类

StringUitl 类中主要实现了字符串与其他数据类型的转换，例如，将日期时间型数据转换为指定格式的字符串，处理订单号码的生成以及验证字符串和浮点数的有效性。该类中声明的所有方法都是静态方法，以便在其他类中可以通过 StringUitl 类名直接调用。

在方法中通过 new Date()方法获取当前的系统时间，通过 SimpleDateFormat 的 format()方法将日期格式转换为指定的日期格式。该方法主要是在操作数据库的时候作为一个有效字段使用，例如订单的创建日期等，其代码如下。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\util\StringUitl

```
public static String getStringTime(){
    Date date = new Date();                                         //获取当前系统时间
SimpleDateFormat sdf = new SimpleDateFormat("yyyyMMddHHmmssSSSS");  //设置格式化格式
    return sdf.format(date);                                        //返回格式化后的时间
}
```

23.5.5　实体映射

由于本程序中使用了 Hibernate 框架，所以需要创建实体对象并通过 Hibernate 的映射文件将实体对象与数据库中相应的数据表进行关联。在 Go 购商城中有 5 个主要的实体对象，分别是会员实体对象、订单实体对象、订单条目实体对象、商品实体对象以及商品类别实体对象。

1. 实体对象总体设计

实体对象是 Hibernate 中非常重要的一个环节，因为 Hibernate 只有通过映射文件建立实体对象与数据库数据表之间的关系，才能进行系统的持久化操作。在 Go 购商城网站中主要实体对象及其关系如图 23.13 所示。

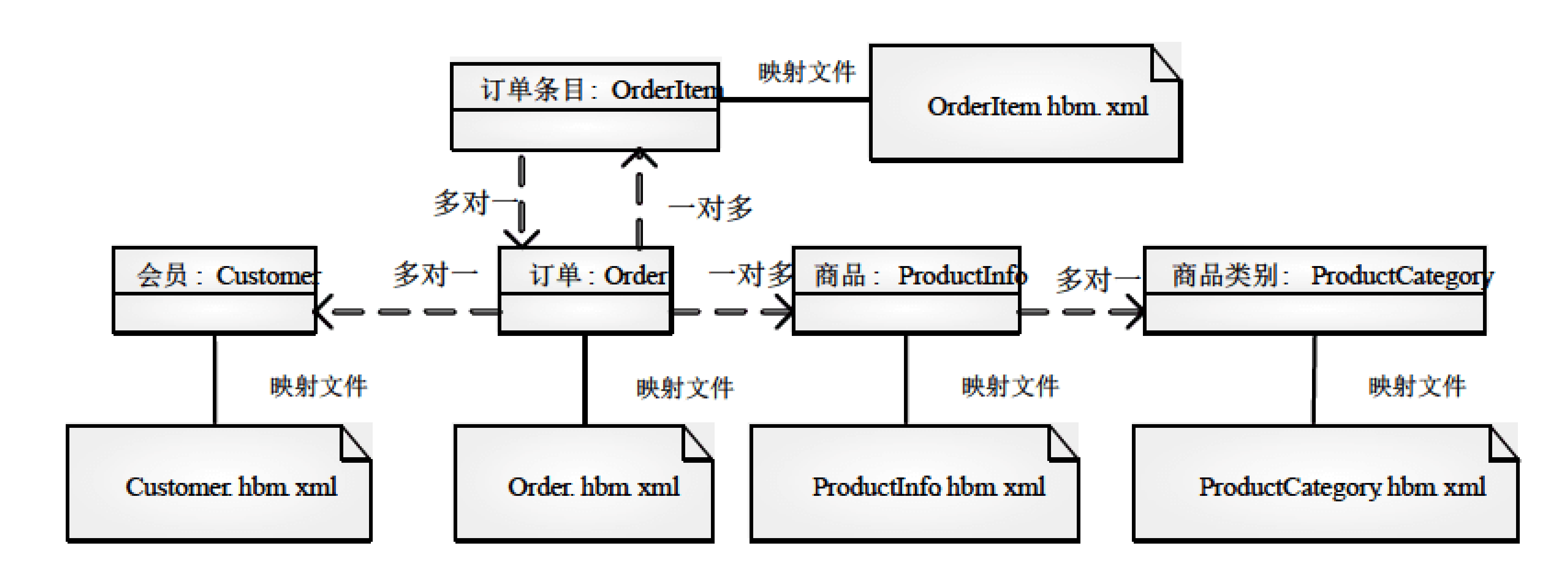


图 23.13　Go 购商城主要实体对象及其关系

从图 23.13 中可以看到，该项目主要有 5 个实体对象，分别是会员实体对象 Customer 类、订单实体对象 Order 类、订单条目实体对象 OrderItem 类、商品实体对象 ProductInfo 类和商品类别实体对象

ProductCategory 类。

从中可以看到会员与订单是一对多的关系，一个会员可以对应多张订单，但是每张订单只能对应一个会员；订单条目与订单为多对一的关系，一张订单中可以包含多个订单条目，但是每个订单条目只能对应一张订单；订单与产品是一对多关系，一张订单可以对应多个商品；商品与商品类别是多对一关系，多件商品可以对应一个商品类别。

其中的“*.hbm.xml”文件为实体对象的 Hibernate 映射文件。

2. 会员信息

Customer 类为会员信息实体类，用于封装会员的注册信息，其关键代码如下。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\user\Customer

```
public class Customer implements Serializable{
	private Integer id;				//用户编号
	private String username;			//用户名
	private String password;			//密码
	private String realname;			//真实姓名
	private String email;				//邮箱
	private String address;			//住址
	private String mobile;				//手机
	…						//省略的 Setter 和 Getter 方法
}
```

创建会员信息实体类的映射文件 Customer.hbm.xml，在映射文件中配置会员实体类属性与数据表 tb_customer 响应字段的关联，并声明用户编号的主键生成策略为自动增长，配置文件中的关键代码如下。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\user\Customer.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
	"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
	"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd" >
<hibernate-mapping package="com.lyq.model.user">
	<class name="Customer" table="tb_customer">
	<id name="id">
	<generator class="native"/>
	</id>
	<property name="username" not-null="true" length="50"/>
	<property name="password" not-null="true" length="50"/>
	<property name="realname" length="20"/>
	<property name="address" length="200"/>
	<property name="email" length="50"/>
	<property name="mobile" length="11"/>
	</class>
</hibernate-mapping>
```

3. 订单信息

Order 类为订单信息实体类，用户封装订单的基本信息，但是不包括详细的订购信息，其关键代码如下。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\model\order\Order

```
public class Order implements Serializable {
	private String orderId;				//订单编号(手动分配)
	private Customer customer;			//所属用户
	private String name;				//收货人姓名
	private String address;				//收货人住址
	private String mobile;				//收货人手机
	private Set<OrderItem> orderItems;		//所买商品
	private Float totalPrice;			//总额
	private PaymentWay paymentWay;			//支付方式
	private OrderState orderState;			//订单状态
	private Date createTime = new Date();		//创建时间
	…						//省略的 Setter 和 Getter 方法
}
```

创建订单信息实体类的映射文件 Order.hbm.xml，在映射文件中配置订单实体类属性与数据表 tb_order 字段的关联，声明主键 orderId 的主键生成策略为手动分配，并配置订单与会员的多对一关系，订单与订单项的一对多关系，其关键代码如下。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\model\order\Order

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
	"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
	"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd" >
<hibernate-mapping package="com.lyq.model.order">
	<class name="Order" table="tb_order">
	<id name="orderId" type="string" length="30">
	<generator class="assigned"/>
	</id>
	<property name="name" not-null="true" length="50"/>
	<property name="address" not-null="true" length="200"/>
	<property name="mobile" not-null="true" length="11"/>
	<property name="totalPrice"/>
	<property name="createTime" />
	<property name="paymentWay" type="com.lyq.util.hibernate.PaymentWayType" length="15"/>
	<property name="orderState" type="com.lyq.util.hibernate.OrderStateType" length="10"/>
	<!-- 多对一映射用户 -->
	<many-to-one name="customer" column="customerId"/>
	<!-- 映射订单项 -->
	<set name="orderItems" inverse="true" lazy="extra" cascade="all">
	<key column="orderId"/>
	<one-to-many class="OrderItem"/>
	</set>
	</class>
</hibernate-mapping>
```

4. 订单条目信息

OrderItem 类为订单条目的实体对象，用于封装一个订单中的一条详细商品采购信息，其关键代码如下。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\model\order\OrderItem

```
public class OrderItem implements Serializable{
      private Integer id;                              //商品条目编号
      private Integer productId;                       //商品 id
      private String productName;                      //商品名称
      private Float productMarketprice;                //市场价格
      private Float productPrice;                      //商品销售价格
      private Integer amount=1;                        //购买数量
      private Order order;                             //所属订单 iD
      …                                                //省略的 Setter 和 Getter 方法
}
```

创建订单条目信息实体类的映射文件 OrderItem.hbm.xml，在映射文件中配置订单条目实体类属性与数据表 tb_orderitem 字段的关联，声明主键 id 的主键生成策略为自动增长，并配置订单条目与订单的多对一关系，其关键代码如下。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\model\order\OrderItem.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
      "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
      "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd" >
<hibernate-mapping package="com.lyq.model.order">
      <class name="OrderItem" table="tb_orderItem">
      <id name="id">
      <generator class="native"/>
      </id>
      <property name="productId" not-null="true"/>
      <property name="productName" not-null="true" length="200"/>
      <property name="productPrice" not-null="true"/>
      <property name="amount"/>
      <!-- 多对一映射订单 -->
      <many-to-one name="order" column="orderId"/>
      </class>
</hibernate-mapping>
```

5. 商品信息

ProductInfo 类为商品信息实体类，主要用于封装商品相关的基本信息，它是整个系统中最为重要的一个实体对象，也是应用最多的一个实体对象，整个网站的业务流程都以商品为核心进行展开，其关键代码如下。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\model\product\ProductInfo

```
public class ProductInfo implements Serializable {
      private Integer id;                              //商品编号
```

```
    private String name;                            //商品名称
    private String description;                     //商品说明
    private Date createTime = new Date();           //上架时间
    private Float baseprice;                        //商品采购价格
    private Float marketprice;                      //现在市场价格
    private Float sellprice;                        //商城销售价格
    private Sex sexrequest   ;                      //所属性别
    private Boolean commend = false;                //是否是推荐商品（默认值为 false）
    private Integer clickcount = 1;                 //访问量（统计受欢迎的程度）
    private Integer sellCount = 0;                  //销售数量（统计热销商品）
    private ProductCategory category;               //所属类别
    private UploadFile uploadFile;                  //上传文件
    …                                               //省略的 Setter 和 Getter 方法
}
```

创建商品信息实体类的映射文件 ProductInfo.hbm.xml，在映射文件中配置商品实体类属性与数据表 tb_productinfo 字段的关联，并声明其主键 id 的生成策略为自动增长，并配置商品与商品类别多对一关联关系、商品与商品上传文件的多对一关联关系，其关键代码如下。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\model\product\ProductInfo.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
    "-//Hibernate/Hibernate Mapping DTD 3.0//EN"
    "http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd" >
<hibernate-mapping package="com.lyq.model.product">
    <class name="ProductInfo" table="tb_productInfo">
    <id name="id">
    <generator class="native"/>
    </id>
    <property name="name" not-null="true" length="100"/>
    <property name="description" type="text"/>
    <property name="createTime"/>
    <property name="baseprice"/>
    <property name="marketprice"/>
    <property name="sellprice"/>
    <property name="sexrequest" type="com.lyq.util.hibernate.SexType" length="5"/>
    <property name="commend"/>
    <property name="clickcount"/>
    <property name="sellCount"/>
    <!-- 多对一映射类别 -->
    <many-to-one name="category" column="categoryId"/>
    <!-- 多对一映射上传文件 -->
    <many-to-one name="uploadFile" unique="true" cascade="all" lazy="false"/>
    </class>
</hibernate-mapping>
```

6. 商品类别信息

ProductCategory 类为商品类别的实体对象，主要用于封装商品类别的基本信息，其关键代码如下。



代码位置：光盘\TM\sl\23\Shop\src\com\lyq\model\product\ProductCategory

```
public class ProductCategory implements Serializable {
	private Integer id;                                                   //类别编号
	private String name;                                                  //类别名称
	private int level = 1;                                                //层次
	private Set<ProductCategory> children;                                //子产品类别
	private ProductCategory parent;                                       //父类别
	private Set<ProductInfo> products = new TreeSet<ProductInfo>();      //包含商品
	…                                                                     //省略的 Setter 和 Getter 方法
}
```

创建商品类别信息实体类的映射文件 ProductCategory.hbm.xml，在映射文件中配置商品类别实体类属性与数据表 tb_productcategory 字段的关联，并配置商品类别与商品的一对多的关联关系，商品类别与其父节点多对一的关联关系，商品类别与其子节点一对多的关联关系，其关键代码如下。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\model\product\ProductCategory.hbm.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-mapping PUBLIC
	"-//Hibernate/Hibernate Mapping DTD 3.0//EN"
	"http://hibernate.sourceforge.net/hibernate-mapping-3.0.dtd" >
<hibernate-mapping package="com.lyq.model.product">
	<class name="ProductCategory" table="tb_productCategory">
	<id name="id">
	<generator class="native"/>
	</id>
	<property name="name" not-null="true" length="50"/>
	<property name="level"/>
	<!-- 映射包含的商品集合 -->
	<set name="products" inverse="true" lazy="extra">
	<key column="categoryId"/>
	<one-to-many class="ProductInfo" />
	</set>
	<!-- 映射父节点 -->
	<many-to-one name="parent" column="pid"/>
	<!-- 映射子节点 -->
	<set name="children" inverse="true" lazy="extra" cascade="all" order-by="id">
	<key column="pid"/>
	<one-to-many class="ProductCategory"/>
	</set>
	</class>
</hibernate-mapping>
```

23.6　项目环境搭建

项目正式开发的第一步就是搭建项目的环境以及项目集成的框架等，都说万丈高楼平地起，从此

开始将踏上万里征程的第一步，在此之前需要将 Spring、Struts 2、Hibernate 以及系统应用的其他 jar 包导入项目的 lib 文件下。

23.6.1 配置 Struts 2

struts.xml 文件是 Struts 2 重要的配置文件，通过对该文件的配置实现程序的 Action 与用户请求之间的映射、视图映射等重要的配置信息。在项目的 ClassPath 下创建 struts.xml 文件，其配置代码如下。

代码位置：光盘\TM\sl\23\Shop\src\struts.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts PUBLIC
      "-//Apache Software Foundation//DTD Struts Configuration 2.1//EN"
      "http://struts.apache.org/dtds/struts-2.1.dtd" >
<struts>
      <!-- 前后台公共视图的映射 -->
      <include file="com/lyq/action/struts-default.xml" />
      <!-- 后台管理的 Struts 2 配置文件 -->
      <include file="com/lyq/action/struts-admin.xml" />
      <!-- 前台台管理的 Struts 2 配置文件 -->
      <include file="com/lyq/action/struts-front.xml" />
</struts>
```

为了便于程序的维护和管理，将前后台的 Struts 2 配置文件进行分开处理，然后通过 include 标签加载在系统默认加载的 Struts 2 配置文件中。在此将 Struts 2 配置文件分为 3 个部分，struts-default.xml 文件为前后台公共的视图映射配置文件，其代码如下。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\action\struts-default.xml

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE struts PUBLIC
     "-//Apache Software Foundation//DTD Struts Configuration 2.1//EN"
     "http://struts.apache.org/dtds/struts-2.1.dtd">
<struts>
      <!-- OGNL 可以使用静态方法 -->
      <constant name="struts.ognl.allowStaticMethodAccess" value="true"/>
      <package name="shop-default" abstract="true" extends="struts-default">
      <global-results>
                 …<!--省略的配置信息 -->
           </global-results>
      <global-exception-mappings>
      <exception-mapping result="error"
      exception="com.lyq.util.AppException"></exception-mapping>
      </global-exception-mappings>
      </package>
</struts>
```

后台管理的 Struts 2 配置文件 struts-admin.xml 主要负责后台用户请求的 Action 和视图映射，其代码如下。



代码位置：光盘\TM\sl\23\Shop\src\com\lyq\action\struts-admin.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts PUBLIC
        "-//Apache Software Foundation//DTD Struts Configuration 2.1//EN"
        "http://struts.apache.org/dtds/struts-2.1.dtd" >
<struts>
        <!-- 后台管理 -->
        <package name="shop.admin" namespace="/admin" extends="shop-default">
                <!-- 配置拦截器 -->
                <interceptors>
                        <!-- 验证用户登录的拦截器 -->
                        <interceptor name="loginInterceptor"
                                class="com.lyq.action.interceptor.UserLoginInterceptor"/>
                        <interceptor-stack name="adminDefaultStack">
                                <interceptor-ref name="loginInterceptor"/>
                        <interceptor-ref name="defaultStack"/>
                        </interceptor-stack>
                </interceptors>
                <action name="admin_*" class="indexAction" method="{1}">
                        <result name="top">/WEB-INF/pages/admin/top.jsp</result>
                                …<!--省略的 Action 配置 -->
                                <interceptor-ref name="adminDefaultStack"/>
                </action>
        </package>
        <package name="shop.admin.user" namespace="/admin/user" extends="shop-default">
                <action name="user_*" method="{1}" class="userAction"></action>
        </package>
       <!-- 栏目管理 -->
       <package name="shop.admin.category" namespace="/admin/product" extends="shop.admin">
                <action name="category_*" method="{1}" class="productCategoryAction">
                        …<!--省略的 Action 配置 -->
                        <interceptor-ref name="adminDefaultStack"/>
                </action>
       </package>
       <!-- 商品管理 -->
       <package name="shop.admin.product" namespace="/admin/product" extends="shop.admin">
                <action name="product_*" method="{1}" class="productAction">
                        …<!--省略的 Action 配置 -->
                        <interceptor-ref name="adminDefaultStack"/>
                </action>
       </package>
       <!-- 订单管理 -->
       <package name="shop.admin.order" namespace="/admin/product" extends="shop.admin">
                <action name="order_*" method="{1}" class="orderAction">
                        …<!--省略的 Action 配置 -->
                        <interceptor-ref name="adminDefaultStack"/>
                </action>
       </package>
</struts>
```

前台管理的 Struts 2 配置文件 struts-front.xml 主要负责后台用户请求的 Action 和视图映射，其代码如下。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\action\struts-front.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE struts PUBLIC
        "-//Apache Software Foundation//DTD Struts Configuration 2.1//EN"
        "http://struts.apache.org/dtds/struts-2.1.dtd" >
<struts>
        <!-- 程序前台 -->
        <package name="shop.front" extends="shop-default">
                <!-- 配置拦截器 -->
                <interceptors>
                        <!-- 验证用户登录的拦截器 -->
                        <interceptor name="loginInterceptor"
                                class="com.lyq.action.interceptor.CustomerLoginInteceptor"/>
                        <interceptor-stack name="customerDefaultStack">
                                <interceptor-ref name="loginInterceptor"/>
                        <interceptor-ref name="defaultStack"/>
                        </interceptor-stack>
                </interceptors>
                <action name="index" class="indexAction">
                    <result>/WEB-INF/pages/index.jsp</result>
              </action>
        </package>
        <!-- 消费者 Action -->
        <package name="shop.customer" extends="shop-default" namespace="/customer">
                <action name="customer_*" method="{1}" class="customerAction"></action>
        </package>
        <!-- 商品 Action -->
        <package name="shop.product" extends="shop-default" namespace="/product">
                <action name="product_*" class="productAction" method="{1}">
                        …<!--省略的 Action 配置 -->
                </action>
        </package>
        <!-- 购物车 Action -->
        <package name="shop.cart" extends="shop.front" namespace="/product">
                <action name="cart_*" class="cartAction" method="{1}">
                        …<!--省略的 Action 配置 -->
                        <interceptor-ref name="customerDefaultStack"/>
                </action>
        </package>
        <!-- 订单 Action -->
        <package name="shop.order" extends="shop.front" namespace="/product">
                <action name="order_*" class="orderAction" method="{1}">
                        …<!--省略的 Action 配置 -->
                        <interceptor-ref name="customerDefaultStack"/>
                </action>
        </package>
</struts>
```

Struts 2 与 Struts 1 是两个完全不同的开发框架，除其核心控制器不同以外，还有以下 3 个不同点。

1. 命名空间的不同

在 Struts 2 中的 Filter 默认的扩展名为.action，这是在 default.properties 文件的 struts.action.extension 属性中定义的；而 Struts 1 则通过<init-pattern>属性来配置。Struts 1 与 Struts 2 两个框架的命名空间不一样，因此 Struts 1 和 Struts 2 可以在同一个 Web 应用系统中无障碍地共存。

2. 设置系统属性的不同

Struts 2 的配置信息中不需要通过<init-param>来设置系统的属性，它并不是取消了这些属性，而是使用 default.properties 文件作为默认的配置选项文件，并可以通过 struts.properties 的文件设置不同的属性值来覆盖默认文件的值实现自己的配置。

3. 映射文件名的配置参数的不同

在 Struts 2 中没有提供映射文件名的配置参数，取而代之的是默认配置文件 struts.xml。

23.6.2　配置 Hibernate

Hibernate 配置文件主要用于配置数据库连接和 Hibernate 运行时所需的各种属性，这个配置文件位于应用程序或 Web 程序的类文件夹 classes 中。Hibernate 配置文件支持两种形式，一种是 XML 格式的配置文件；另一种是 Java 属性文件格式的配置文件，采用“键=值”的形式。建议采用 XML 格式的配置文件。

在 Hibernate 的配置文件中配置连接的数据库的连接信息，数据库方言以及打印 SQL 语句等属性，其关键代码如下。

代码位置：光盘\TM\sl\23\Shop\src\hibernate.cfg.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hibernate-configuration PUBLIC
	"-//Hibernate/Hibernate Configuration DTD 3.0//EN"
	"http://hibernate.sourceforge.net/hibernate-configuration-3.0.dtd" >
<hibernate-configuration>
	<session-factory>
		<!-- 数据库方言 -->
		<property name="hibernate.dialect">org.hibernate.dialect.MySQLDialect</property>
		<!-- 数据库驱动 -->
		<property name="hibernate.connection.driver_class">com.mysql.jdbc.Driver</property>
		<!-- 数据库连接信息 -->
		<property name="hibernate.connection.url">
		jdbc:mysql://localhost:3306/db_database24</property>
		<property name="hibernate.connection.username">root</property>
		<property name="hibernate.connection.password">111</property>
		<!-- 打印 SQL 语句 -->
		<property name="hibernate.show_sql">true</property>
		<!-- 不格式化 SQL 语句 -->
		<property name="hibernate.format_sql">false</property>
```

```
            <!-- 为 Session 指定一个自定义策略 -->
            <property name="hibernate.current_session_context_class">thread</property>
            <!-- C3P0 JDBC 连接池 -->
            <property name="hibernate.c3p0.max_size">20</property>
            <property name="hibernate.c3p0.min_size">5</property>
            <property name="hibernate.c3p0.timeout">120</property>
            <property name="hibernate.c3p0.max_statements">100</property>
            <property name="hibernate.c3p0.idle_test_period">120</property>
            <property name="hibernate.c3p0.acquire_increment">2</property>
            <property name="hibernate.c3p0.validate">true</property>
            <!-- 映射文件 -->
            <mapping resource="com/lyq/model/user/User.hbm.xml"/>
            …<!--省略的映射文件 -->
        </session-factory>
</hibernate-configuration>
```

说明

C3P0 是一个随 Hibernate 一同分发的开发的 JDBC 连接池，它位于 Hibernate 源文件的 lib 目录下。如果在配置文件中设置了 hibernate.c3p0.*的相关属性，Hibernate 将会使用 C3P0Connection Provider 来缓存 JDBC 连接。

23.6.3 配置 Spring

利用 Spring 加载 Hibernate 的配置文件以及 Session 管理类，所以在配置 Spring 的时候，只需要配置 Spring 的核心配置文件 applicationContext-common.xml，其代码如下。

代码位置：光盘\TM\sl\23\Shop\src\applicationContext-common.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:context="http://www.springframework.org/schema/context"
        xmlns:aop="http://www.springframework.org/schema/aop"
        xmlns:tx="http://www.springframework.org/schema/tx"
        xsi:schemaLocation="http://www.springframework.org/schema/beans
                http://www.springframework.org/schema/beans/spring-beans-2.19.xsd
                http://www.springframework.org/schema/context
        http://www.springframework.org/schema/context/spring-context-2.19.xsd
                http://www.springframework.org/schema/aop
        http://www.springframework.org/schema/aop/spring-aop-2.19.xsd
                http://www.springframework.org/schema/tx
        http://www.springframework.org/schema/tx/spring-tx-2.19.xsd">
        <context:annotation-config/>
        <context:component-scan base-package="com.lyq"/>
        <!-- 配置 sessionFactory -->
        <bean id="sessionFactory"
            class="org.springframework.orm.hibernate3.LocalSessionFactoryBean">
```

```
            <property name="configLocation">
                <value>classpath:hibernate.cfg.xml</value>
            </property>
        </bean>
        <!-- 配置事务管理器 -->
        <bean id="transactionManager"
            class="org.springframework.orm.hibernate3.HibernateTransactionManager">
            <property name="sessionFactory">
                <ref bean="sessionFactory" />
            </property>
        </bean>
        <tx:annotation-driven transaction-manager="transactionManager" />
        <!-- 定义 Hibernate 模板对象 -->
        <bean id="hibernateTemplate" class="org.springframework.orm.hibernate3.HibernateTemplate">
            <property name="sessionFactory" ref="sessionFactory"/>
        </bean>
</beans>
```

23.6.4　配置 web.xml

任何 MVC 框架都需要与 Servlet 应用整合，而 Servlet 则必须在 web.xml 文件中进行配置。web.xml 的配置文件是项目的基本配置文件，通过该文件设置实例化 Spring 容器、过滤器、配置 Struts 2 以及设置程序默认执行的操作，其关键代码如下。

代码位置：光盘\TM\sl\23\Shop\WebContent\WEB-INF\web.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xmlns:web="http://java.sun.com/xml/ns/javaee/web-app_2_19.xsd"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
    http://java.sun.com/xml/ns/javaee/web-app_2_19.xsd"
    id="WebApp_ID" version="2.5">
    <display-name>Shop</display-name>
    <!-- 对 Spring 容器进行实例化 -->
    <listener>
    <listener-class>org.springframework.web.context.ContextLoaderListener</listener-class>
    </listener>
    <context-param>
    <param-name>contextConfigLocation</param-name>
    <param-value>classpath:applicationContext-*.xml</param-value>
    </context-param>
    <!-- OpenSessionInViewFilter 过滤器 -->
    <filter>
    <filter-name>openSessionInViewFilter</filter-name>
    <filter-class>org.springframework.orm.hibernate3.support.OpenSessionInViewFilter</filter-class>
    </filter>
    <filter-mapping>
    <filter-name>openSessionInViewFilter</filter-name>
```

```
        <url-pattern>/*</url-pattern>
        </filter-mapping>
        <!—Struts 2 配置 -->
        <filter>
            <filter-name>struts2</filter-name>
            <filter-class>org.apache.struts2.dispatcher.ng.filter.StrutsPrepareAndExecuteFilter</filter-class>
       </filter>
       <filter-mapping>
            <filter-name>struts2</filter-name>
            <url-pattern>/*</url-pattern>
        </filter-mapping>
        <!-- 设置程序的默认欢迎页面-->
        <welcome-file-list>
        <welcome-file>index.jsp</welcome-file>
        </welcome-file-list>
</web-app>
```

23.7 登录注册模块设计

如果要提高网站的安全性，防止非法用户进入网站，可以让用户进入网站前先进行注册，注册成功的用户才可以通过购物车购买商品。用户注册在大多数网站中都是不可缺少的功能，也是用户参与网站活动最为直接的桥梁。通过用户注册，可以有效地对用户信息进行采集，并将合法的用户信息保存到指定的数据表中，通常情况下，当用户注册操作完毕，将直接登录该网站。

23.7.1 模块概述

由于 Go 购商城分为前台和后台两个部分，所以登录也分为前台登录和后台登录两个部分功能，前台的登录针对的是在 Go 购商城注册的会员，后台登录主要针对的是网站管理员，而注册模块主要针对的就是前台想进行购物的游客，如图 23.14 所示。

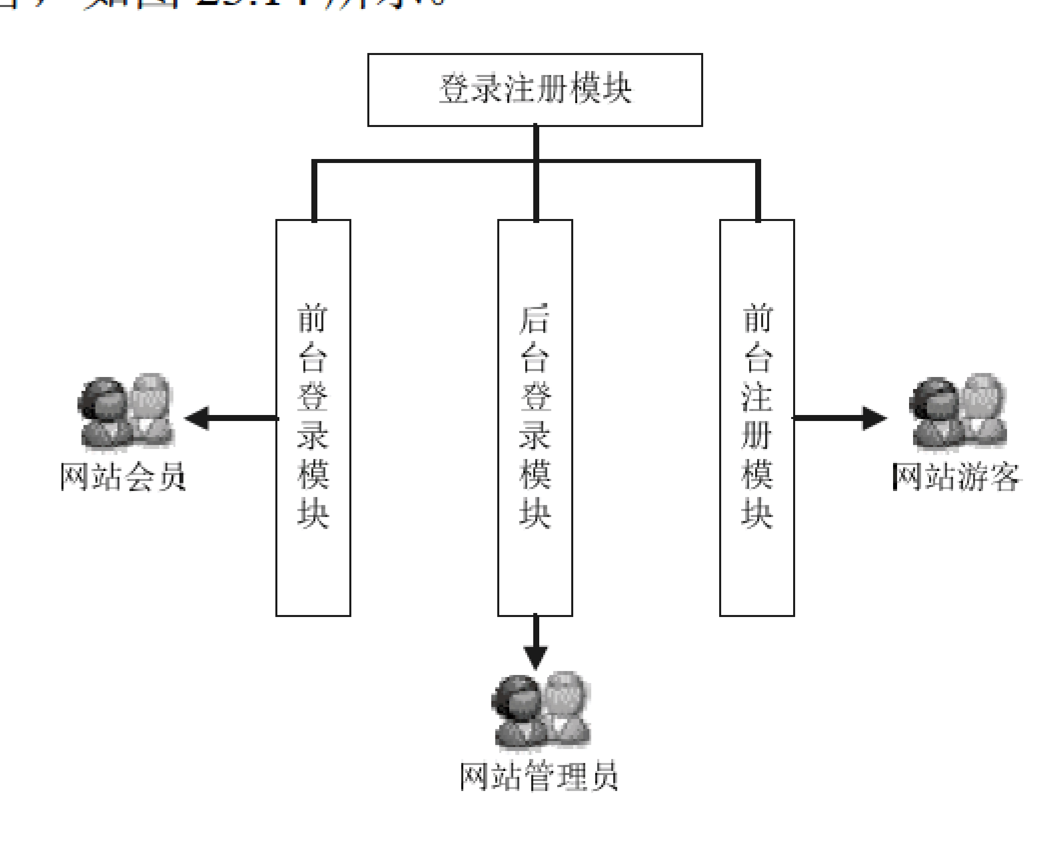


图 23.14 登录注册模块框架图

23.7.2　注册模块的实现

在安全注册与登录操作过程中，主要是将表单内容进行严格的校验，这样可以提高网站的安全性，防止非法用户进入网站。

在本模块中，用户注册页面为 customer_reg.jsp，如图 23.15 所示，用户注册主要包括用户名、密码、确认密码、邮箱地址、住址以及手机号码 6 个表单控件。其中，用户名要求 5～32 个字符，密码表单与确认密码表单必须一致；邮箱地址表单必须是正确的地址，这里通过 Struts 2 的校验器进行校验；住址与手机号码两个表单主要是在用户购买商品生成订单时直接获取相关的送货信息，方便用户的操作。

当用户成功注册信息后，就可以进行登录操作，并在 Go 购商城中购买商品。



图 23.15　会员注册页面

1. 表单验证

在 Web 开发中，为了确保数据的安全性，通常情况下都会对页面提交的数据进行统一验证，从而保障数据的合法性。

在 Struts 2 中有两种表单的验证方式，在本模块中将使用 XML 文件对表单中的信息进行合法性验证，利用 requiredsting 校验器对 CustomerAction 类中的字段进行非空验证；利用 stringlength 校验器对 CustomerAction 类中的字段长度进行验证，利用 email 校验器对邮箱地址的格式进行验证，在 CustomerAction 类的包下新建 XML 文件。

2. 保存注册信息

当用户单击会员注册页面中的“注册”超链接时，系统将会发送一个 customer_reg.html 的 URL 请求，该请求将会执行 CustomerAction 类中的 save()方法，在该方法中首先判断用户名是否可用，如果可用就将注册信息保存在数据库中，否则返回错误信息，其关键代码如下。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\action\user\CustomerAction

```
public String save() throws Exception{
      boolean unique = customerDao.isUnique(customer.getUsername());      //判断用户名是否可用
      if(unique){                                                         //如果用户名可用
      customerDao.save(customer);                                         //保存注册信息
      return CUSTOMER_LOGIN;                                              //返回会员登录页面
      }else{
      throw new AppException("此用户名不可用");                             //否则返回页面错误信息
      }
}
```

23.8　前台商品信息查询模块设计

商品是 Go 购商城的灵魂，只有好的商品展示以及丰富的商品信息才能吸引顾客的眼球，提高网站的关注度，这也是为企业创造效益的决定性因素，所以 Go 购商城的前台商品展示在整个系统中占有非常重要的地位。

23.8.1　模块概述

根据前台的页面设计将前台商品信息查询模块划分为 5 个模块，主要包括商品分类查询、人气商品查询、热销商品查询、推荐商品查询以及商品的模糊查询，如图 23.16 所示。

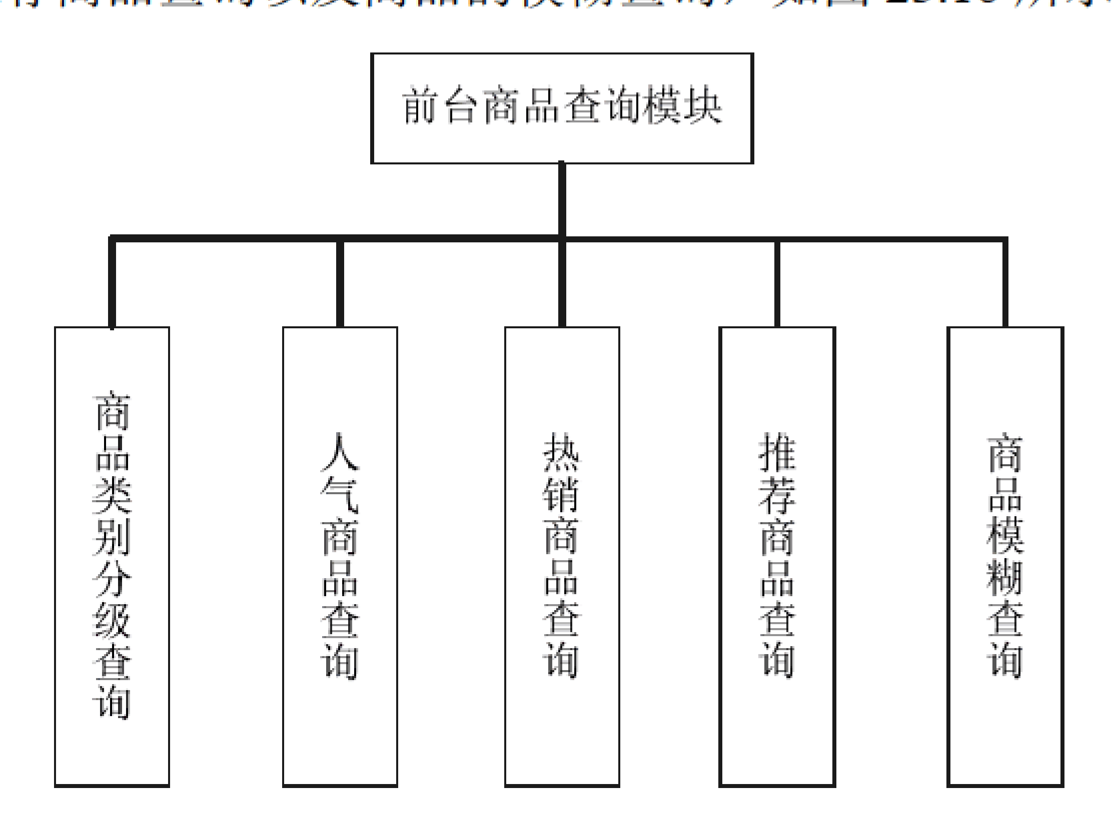


图 23.16　前台商品信息查询模块框架图

23.8.2　前台商品信息查询模块技术分析

在前台的首页商品展示中，首先展现给用户的就是商品类别的分级显示，方便用户按类别对商品进行查询，同时也能体现出 Go 购商城产品种类的丰富多样。

实现商品类别的分级查询，首先需要查询所有的一级节点，通过公共模块持久化类中封装的 find()方法实现该功能，在首页的 Action 请求 IndexAction 的 execute()方法中，调用封装的 find()方法，其关键代码如下。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\action\IndexAction

```
public String execute() throws Exception {
	//查询所有类别
	String where = "where parent is null";
	categories = categoryDao.find(-1, -1, where, null).getList();
```



```
    …                          //省略的 Setter 和 Getter 方法
}
```

在 find()方法中含有 4 个参数，其中，-1 参数分别代表当前页数和每页显示的记录数，where 参数代表的是查询条件，null 参数代表的是数据排序的条件参数。find()方法会根据提供的两个-1 参数执行以下代码。

```
//如果 maxResult<0，则查询所有
if(maxResult < 0 && pageNo < 0){
    list = query.list();                //将查询结果转化为 List 对象
}
```

23.8.3　商品搜索的实现

在 Go 购商城中主要实现普通搜索，在对数据表的简单搜索中，当搜索表单中没有输入任何数据时，单击“搜索”按钮后，可以对数据表中的所有内容进行查询；当在关键字文本框中输入要搜索的内容，单击“搜索”按钮后，可以按关键字内容查询数据表中所有的内容。该功能方便了用户对商品信息的查找，用户可以在首页的文本输入框中输入关键字搜索指定的商品信息，如图 23.17 所示。



图 23.17　商品搜索的效果

商品搜索的方法封装在 ProductAction 类中，通过 HQL 的 like 条件语句实现商品的模糊查询的功能，其关键代码如下。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\action\product\ProductAction

```
public String findByName() throws Exception {
    if(product.getName() != null){
```

```
        String where = "where name like ?";                                  //查询的条件语句
        Object[] queryParams = {"%" + product.getName() + "%"};                //为参数赋值
        pageModel = productDao.find(pageNo, pageSize, where, queryParams );   //执行查询方法
        }
        return LIST;                                                          //返回列表首页
}
```

在商品的列表页面中，通过 Struts 2 的<s:iterator>标签遍历返回的商品 List 集合，其关键代码如下。

代码位置：光盘\TM\sl\23\Shop\WebContent\WEB-INF\pages\product\product_list.jsp

```
<s:iterator value="pageModel.list">
        <ul>
                <li>
                <table border="0" width="100%" cellpadding="0" cellspacing="0">
                        <tr>
                                <td rowspan="5" width="160">
                                        <s:a action="product_select" namespace="/product">
                                        <s:param name="id" value="id"></s:param>
                                        <img width="150" height="150"src="<s:property
                                        value="#request.get('javax.servlet.forward.context_path')"/>/upload
                                        /<s:property value="uploadFile.path"/>">
                                </s:a></td>
                        </tr>
                        <tr bgcolor="#f2eec9">
                                <td align="right" width="90">商品名称：</td>
                                <td><s:a action="product_select" namespace="/product">
                                        <s:param name="id" value="id"></s:param>
                                        <s:property value="name" />
                                </s:a></td>
                        </tr>
                        <tr>
                                <td align="right" width="90">市场价格：</td>
                                <td><font style="text-decoration: line-through;">
                                <s:property value="marketprice" /> </font></td>
                        </tr>
                        <tr bgcolor="#f2eec9">
                                <td align="right" width="90">Go 购价格：</td>
                                <td><s:property value="sellprice" />
                                        <s:if test="sellprice <= marketprice">
                                        <font color="red">节省
                                        <s:property value="marketprice-sellprice" /></font>
                                </s:if></td>
                        </tr>
                        <tr>
                                <td colspan="2" align="right">
                                        <s:a action="product_select" namespace="/product">
                                        <s:param name="id" value="id"></s:param>
                                        <img src="${context_path}/css/images/gm_06.gif" width="136"
                                                height="32" />
                                </s:a></td>
```

```
                </tr>
            </table>
            </li>
        </ul>
</ s:iterator >
```

23.8.4　前台商品其他查询的实现

人气商品推荐模块（如图 23.18 所示）、推荐商品模块（如图 23.19 所示）和热销商品模块（如图 23.20 所示）3 个查询方式的实现方式基本相同，都是通过条件语句进行排序查询，只不过查询的条件不同。



图 23.18　人气商品模块



图 23.19　推荐商品模块



图 23.20　热销商品模块

1. 人气商品模块的实现

人气商品的定义是按照商品的浏览量最多进行定义的，系统将筛选出商品中浏览量最多的几件商品进行展示，商品结果集中的信息将按照商品浏览量进行倒序排列，其实现的方法封装在 ProductAction 类中，其关键代码如下。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\action\product\ProductAction

```
public String findByClick() throws Exception{
    Map<String, String> orderby = new HashMap<String, String>();      //定义 Map 集合
    orderby.put("clickcount", "desc");                                 //为 Map 集合赋值
    pageModel = productDao.find(1, 8, orderby );                       //执行查找方法
    return "clickList";                                                //返回 product_click_list.jsp 页面
}
```

在调用的 find()方法中使用了 3 个参数，前两个参数设置显示的起始位置和显示记录数，最后一个参数是商品信息排序的条件，程序最终返回到 product_click_list.jsp 页面，代码如下。

代码位置：光盘\TM\sl\23\Shop\WebContent\WEB-INF\pages\product\product_list.jsp

```
<s:set var="context_path" value="#request.get('javax.servlet.forward.context_path')"></s:set >
<table width="193" height="23" border="0" cellpadding="0" cellspacing="0">
    <s:iterator value="pageModel.list">
    <tr>
```

```
        <td width="187" valign="middle">
        <img src="${context_path}/css/images/h_32.gif" width="20" height="17" />
        <s:a action="product_select" namespace="/product">
        <s:param name="id" value="id"></s:param>
        <s:property value="name"/>（人气：
        <span class="red"><s:property value="clickcount"/></span>）
        </s:a>
        </td>
        </tr>
</s:iterator>
</table>
```

2. 推荐商品和热销商品模块的实现

推荐商品和热销商品模块的实现与商品搜索模块的实现比较类似，都是通过 hql 的排序语句实现的，推荐商品为商品推荐字段 commend 为 true 的商品，并且按商品销量的倒序排列，其实现的关键代码如下。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\action\product\ProductAction

```
public String findByCommend() throws Exception{
        Map<String, String> orderby = new HashMap<String, String>();            //定义 Map 集合
        orderby.put("sellCount", "desc");                                      //为 Map 集合赋值
        String where = "where commend = ?";                                    //设置条件语句
        Object[] queryParams = {true};                                         //设置参数值
        pageModel = productDao.find(where, queryParams, orderby, pageNo, pageSize);    //执行查询方法
        return "findList";                                                     //返回推荐商品页面
}
```

热销商品为销售量较多的商品，只需按照商品销量的倒序进行排列即可，并以分页的方式取出前 6 条信息，其实现的关键代码如下。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\action\product\ProductAction

```
public String findBySellCount() throws Exception{
        Map<String, String> orderby = new HashMap<String, String>();       //定义 Map 集合
        orderby.put("sellCount", "desc");                                 //为 Map 集合赋值
        pageModel = productDao.find(1, 6, orderby );                      //执行查询方法
        return "findList";                                                //返回热销商品页面
}
```

两个模块在页面中展示的信息方式相同，仅以推荐产品为例，其代码如下。

代码位置：光盘\TM\sl\23\Shop\WebContent\WEB-INF\pages\product\product_list.jsp

```
<s:set var="context_path"
        value="#request.get('javax.servlet.forward.context_path')"></s:set>
<div style="width: 195px;">
        <s:iterator value="pageModel.list">
        <div style="float: left; width:45%; text-align: center;">
        <s:a action="product_select" namespace="/product">
        <s:param name="id" value="id"></s:param>
        <img width="90" height="90" border="1" src="<s:propert value="%{context_path}"/>
```



```
            /upload/<s:property value="uploadFile.path"/>">
        <p style="width: 80px;"><s:property value="name"/></p>
        </s:a>
        </div>
        </s:iterator>
</div>
```

在 Struts 2 的前台 Action 配置文件 struts-front.xml 中，配置前台商品管理模块的 Action 以及视图映射关系，关键代码如下。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\action\struts-front.xml

```
<!-- 商品 Action -->
<package name="shop.product" extends="shop-default" namespace="/product">
        <action name="product_*" class="productAction" method="{1}">
        <result name="list">/WEB-INF/pages/product/product_list.jsp</result>
        <result name="select">/WEB-INF/pages/product/product_select.jsp</result>
        <result name="clickList">/WEB-INF/pages/product/product_click_list.jsp</result>
        <result name="findList">/WEB-INF/pages/product/product_find_list.jsp</result>
        </action>
</package>
```

23.9　购物车模块设计

购物车是商务网站中必不可少的功能，购物车的设计很大程度上会决定网站是否受到用户的关注。商务网站中的购物车会将用户选购的未结算的商品保存一段时间，防止错误操作或意外发生时购物车中的商品丢失，方便了用户的使用。所以在 Go 购商城中购物车也是必不可少的一个模块。

23.9.1　模块概述

Go 购商城购物车实现的主要功能包括添加选购的新商品、自动更新选购的商品数量、清空购物车、自动调整商品总价格以及生成订单信息等。本模块实现的购物车的功能结构如图 23.21 所示。

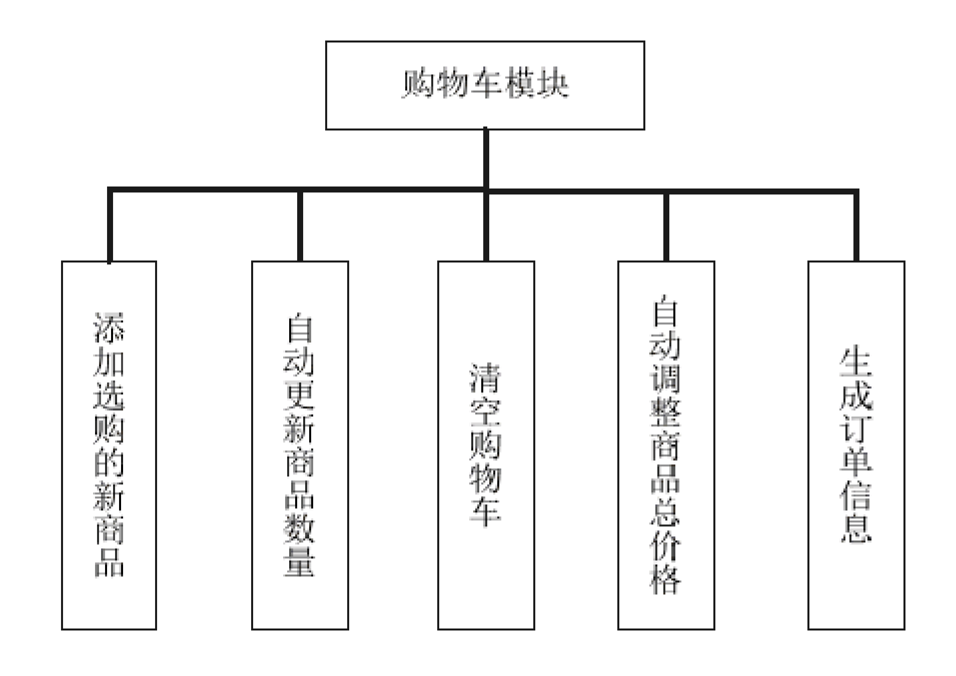


图 23.21　购物车模块的功能结构图

如果用户需要选购商品，必须登录，否则用户无法使用购物车功能。当用户进入购物车后，可以进行结算、清空购物车以及继续选购等操作。当用户进入结算操作后，需要填写订单信息，并选择支付方式，当用户确认支付时系统会生成相应的订单信息。其功能流程图如图 23.22 所示。

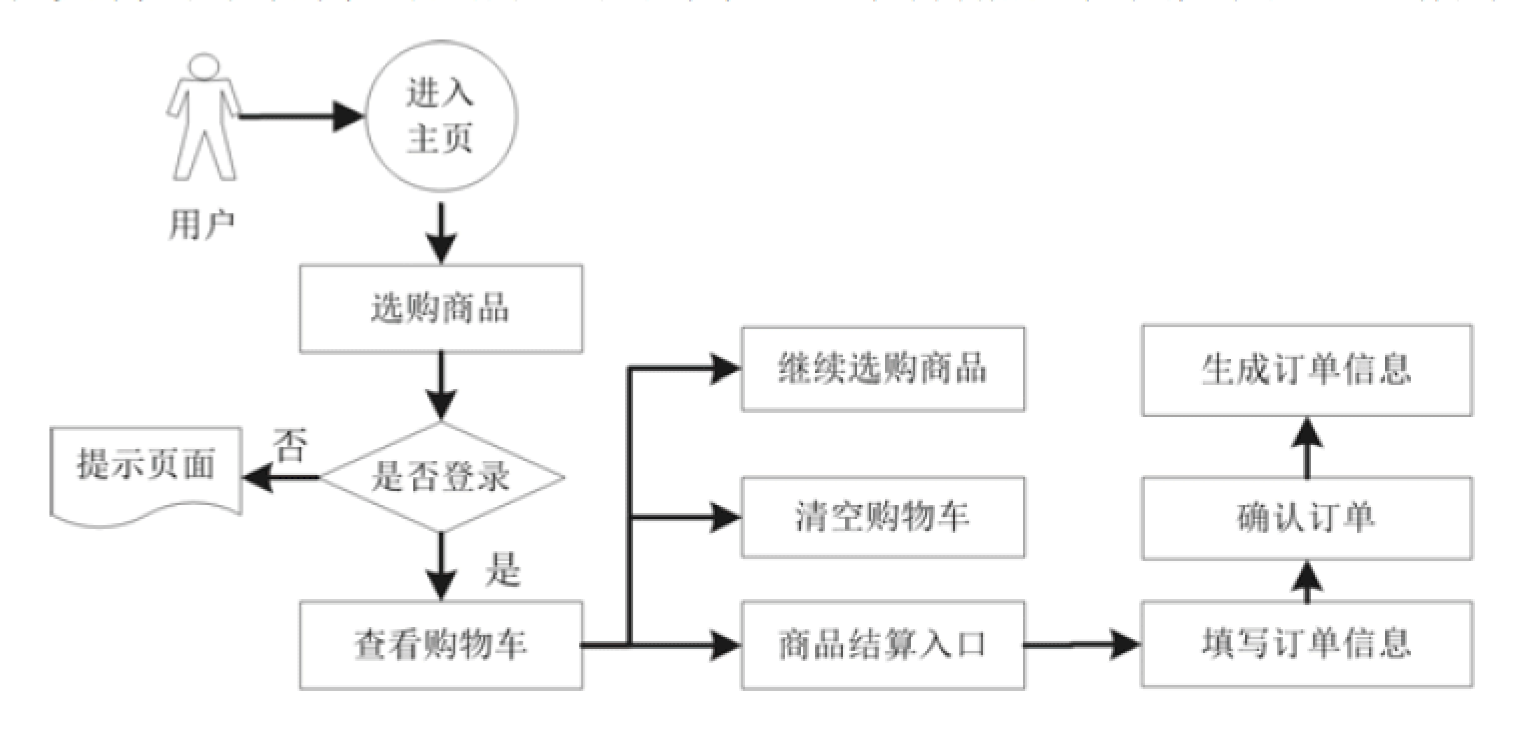


图 23.22　购物车流程图

23.9.2　购物车模块技术分析

在开发时一定要注意有时加入购物车中没有任何的商品采购信息，当用户确认订单的时候，系统同样会生成一个消费金额为 0.0 元且无任何订单条目的订单信息，在系统中该信息是没有任何意义的，而且有可能导致系统不可预知的错误，为了避免这种情况的发生，需要修改前台订单的保存方法，即 OrderAction 类中的 save()方法，判断购物车对象是否为空，如果为空则返回错误信息的提示页面，不进行任何的后续操作。在 save()方法中添加如下代码。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\action\order\OrderAction

```
public String save() throws Exception {
        …                                                   //省略的代码
        Set<OrderItem> cart = getCart();                    //获取购物车
        if(cart.isEmpty()){                                 //判断条目信息是否为空
        return ERROR;                                       //返回订单信息错误提示页面
        }
        …                                                   //省略的代码
}
```

并创建前台订单错误的提示页面 order_error.jsp，当用户误操作导致系统生成的错误订单信息将不会保存到数据库中，而是跳转到错误提示页面。

23.9.3　购物车基本功能的实现

购物车的基本功能包括向购物车中添加商品、清空购物车以及删除购物车中指定的商品订单条目信息 3 项功能，购物车的功能是基于 Session 变量实现的，Session 充当了一个临时信息存储平台，当 Session 失效后，其保存的购物车信息也将全部丢失。其效果图如图 23.23 所示。



图 23.23　购物车内的商品信息

1. 向购物车添加商品

购物车的主要工作就是保存用户的商品购买信息，当登录会员浏览商品详细信息，并单击页面上的“立即购买”超链接时，系统就会将该商品放入购物车内。

在本系统中，将购物车的信息保存在 Session 变量中，其保存的是商品的购买信息，也就是订单的条目信息。所以在向购物车添加商品时，首先要获取商品 ID 进行判断，如果购物车中存在相同的 ID 值，就修改该商品的数量，自动加 1；如果购物车中无相同 ID 的值，则向购物车中添加新的商品购买信息。向购物车添加商品信息的方法封装在 CartAction 类中，其关键代码如下。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\action\order\CartAction

```
public String add() throws Exception {
    if(productId != null && productId > 0){
    Set<OrderItem> cart = getCart();                     //获取购物车
    // 标记添加的商品是否是同一件商品
    boolean same = false;                                //定义 same 布尔变量
    for (OrderItem item : cart) {                        //遍历购物车中的信息
    if(item.getProductId() == productId){
        //购买相同的商品，更新数量
        item.setAmount(item.getAmount() + 1);
        same = true;                                     //设置 same 变量为 true
        }
    }
    //不是同一件商品
    if(!same){
    OrderItem item = new OrderItem();                    //实例化订单条目信息实体对象
    ProductInfo pro = productDao.load(productId);        //加载商品对象
    item.setProductId(pro.getId());                      //设置 id
    item.setProductName(pro.getName());                  //设置商品名称
    item.setProductPrice(pro.getSellprice());            //设置商品销售价格
    item.setProductMarketprice(pro.getMarketprice());    //设置商品市场价格
    cart.add(item);                                      //将信息添加到购物车中
    }
    session.put("cart", cart);                           //将购物车保存在 Session 对象中
    }
    return LIST;
}
```

程序运行结束够将返回订单条目信息的列表页面，即 cart_list.jsp，代码如下。

代码位置：光盘\TM\sl\23\Shop\WebContent\WEB-INF\pages\cart\cart_list.jsp

```
//遍历 Session 对象：通过 Struts 2 的<s:iterator>标签遍历 Session 对象中存放的订单条目信息
<s:iterator value="#session.cart">
	<s:set value="%{#sumall +productPrice*amount}" var="sumall" />
	…<!-- 省略的布局代码 -->
	<td width="213" height="30" align="center">
	<s:property value="productName" /></td>
	<td width="130" align="center">
	<span style="text-decoration: line-through;"> ￥
	<s:property value="productMarketprice" />元</span></td>
	<td width="130" align="center">￥
	<s:property value="productPrice" />元<br>为您节省：￥
//计算“为您节省”金额：其金额的计算公式为（市场价格-销售价格）
	<s:propertyvalue="productMarketprice*amount - productPrice*amount" />元</td>
	<td width="104" align="center" class="red">
		<s:property value="amount" /></td>
	<td width="111" align="center"><s:a action="cart_delete" namespace="/product">
		<s:param name="productId" value="productId"></s:param>
		<img src="${context_path}/css/images/zh03_03.gif" width="52" height="23" />
	</s:a></td>
	…<!-- 省略的布局代码 -->
</s:iterator >
```

2. 删除购物车中指定商品订单条目信息

当用户想删除购物车中的某个商品的订单条目信息时，可以单击信息后的“删除”超链接，就会自动清除该商品的订单条目信息。实现该方法的关键代码如下。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\action\order\CartAction

```
public String delete() throws Exception {
	Set<OrderItem> cart = getCart();			//获取购物车
	//此处使用 Iterator，否则出现 java.util.ConcurrentModificationException
	Iterator<OrderItem> it = cart.iterator();
	while(it.hasNext()){					//使用迭代器遍历商品订单条目信息
	OrderItem item = it.next();
	if(item.getProductId() == productId){
	it.remove();						//移除商品订单条目信息
	}
	}
	session.put("cart", cart);				//将清空后的信息重新放入 Session 中
	return LIST;						//返回购物车页面
}
```

3. 清空购物车

清空购物车的实现较为简单，由于信息是暂时存放于 Session 对象中，所以用户在执行清空购物车操作时，直接清空 Session 对象即可。当用户单击购物车页面中的“清空购物车”超链接时，系统会向服务器发送一个 cart_clear.html 的 URL 请求，该请求执行的是 CartAction 类中的 clear()方法。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\action\order\CartAction

```
public String clear() throws Exception {
        session.remove("cart");                          //移除信息
        return LIST;                                     //返回订单列表页面
}
```

4. 查找购物信息

当用户登录后，可以单击首页顶部的“购物车”链接，查看自己的购物车的相关信息。

当用户单击“购物车”超链接后，系统会发送一个 cart_list.html 的 URL 请求，该请求执行的是 CartAction 中的 list()方法，实现该方法的关键代码如下。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\action\order\CartAction

```
public String list() throws Exception {
        return LIST;                                     //返回购物车页面
}
```

在购物车页面中是通过 Struts 2 的<s:iterator>标签遍历 Session 对象中购物车的相关信息的，在程序模块中并不需要执行任何的操作，只需要返回购物车页面即可。

在 Struts 2 的前台 Action 配置文件 struts-front.xml 中，配置购物车管理模块的 Action 以及视图映射关系，关键代码如下。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\action\struts-front.xml

```
<!-- 购物车 Action -->
<package name="shop.cart" extends="shop.front" namespace="/product">
        <action name="cart_*" class="cartAction" method="{1}">
        <result name="list">/WEB-INF/pages/cart/cart_list.jsp</result>
        <interceptor-ref name="customerDefaultStack"/>
        </action>
</package>
```

23.9.4　订单相关功能的实现

要为选购的商品进行结算，就需要先生成一个订单，订单信息中包括收货人信息、送货方式、支付方式、购买的商品以及订单总价格，当用户在购物车中单击“收银台结账”超链接后，将进入到订单填写的页面，其中包含订单的基本信息，例如收货人姓名、收货人地址、收货人电话以及支付方式，该页面为 order_add.jsp，如图 23.24 所示。下面介绍实现过程。

图 23.24　Go 购商城订单信息添加页面

1. 下订单操作

当用户单击“收银台结账”超链接时，系统将发送一个 order_add.html 的 URL 请求，该请求执行的是 OrderAction 类中的 add()方法，通过该方法将用户的基本信息从 Session 对象中取出，添加到订单表单中指定的位置，并跳转到我的订单页面，其关键代码如下。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\action\order\OrderAction

```
public String add() throws Exception {
    order.setName(getLoginCustomer().getUsername());          //设置收货人姓名
    order.setAddress(getLoginCustomer().getAddress());        //设置收货人地址
    order.setMobile(getLoginCustomer().getMobile());          //设置收货人电话
    return ADD;                                               //返回我的订单页面
}
```

2. 订单确认

在我的订单页面单击“付款”超链接，如图 23.24 所示，将进入订单确认的页面，如图 23.25 所示。在该页面将显示订单的条目信息，也就是用户购买商品的信息清单，以便用户进行确认。



图 23.25　订单确认页面

当用户单击我的订单页面中的“付款”超链接时，系统将发送一个 order_ confirm.html 的 URL 请求，该请求执行的是 OrderAction 类中的 confirm()方法，该方法中只是实现页面的跳转操作，其关键代码如下。



代码位置：光盘\TM\sl\23\Shop\src\com\lyq\action\order\OrderAction

```
public String confirm() throws Exception {
        return "confirm";                                              //返回订单确认页面
}
```

该方法将返回 order_confirm.jsp，该页面即为订单确认页面，其订单条目信息的显示与购物车页面中订单条目信息显示的方法相同。

3. 订单保存

在订单确认页面单击“付款”超链接时，系统将正式生成用户的购物订单，标志着正式的交易开始进行，该链接将会触发 OrderAction 类中的 save()方法，save()方法将把订单信息保存到数据库，其关键代码如下。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\action\order\OrderAction

```
public String save() throws Exception {
        if(getLoginCustomer() != null){                                //如果用户已登录
        order.setOrderId(StringUitl.createOrderId());                  //设置订单号
        order.setCustomer(getLoginCustomer());                         //设置所属用户
        Set<OrderItem> cart = getCart();                               //获取购物车
        //依次将更新订单项中的商品的销售数量
        for(OrderItem item : cart){                                    //遍历购物车中的订单条目信息
        Integer productId = item.getProductId();                       //获取商品 ID
        ProductInfo product = productDao.load(productId);              //装载商品对象
        product.setSellCount(product.getSellCount() + item.getAmount()); //更新商品销售数量
        productDao.update(product);                                    //修改商品信息
        }
        order.setOrderItems(cart);                                     //设置订单项
order.setOrderState(OrderState.DELIVERED);                             //设置订单状态
        float totalPrice = 0f;                                         //计算总额的变量
        for (OrderItem orderItem : cart) {                             //遍历购物车中的订单条目信息
        totalPrice += orderItem.getProductPrice() * orderItem.getAmount(); //商品单价*商品数量
        }
        order.setTotalPrice(totalPrice);                               //设置订单的总价格
        orderDao.save(order);                                          //保存订单信息
        session.remove("cart");                                        //清空购物车
}
        return findByCustomer();                                       //返回消费者订单查询的方法
}
```

执行 save()方法后将返回订单查询的方法 findByCustomer()，在该方法中将以登录用户的 ID 为查询条件，查询该用户的所有订单信息，其关键代码如下。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\action\order\OrderAction

```
public String findByCustomer() throws Exception {
        if(getLoginCustomer() != null){                                //如果用户已登录
        String where = "where customer.id = ?";                        //将用户 id 设置为查询条件
Object[] queryParams = {getLoginCustomer().getId()};                   //创建对象数组
        Map<String, String> orderby = new HashMap<String, String>(1);  //创建 Map 集合
        orderby.put("createTime", "desc");                             //设置排序条件及方式
```

```
        pageModel = orderDao.find(where, queryParams, orderby , pageNo, pageSize);//执行查询方法
        }
        return LIST;                                                        //返回订单列表页面
}
```

该方法将返回用户的订单列表页面 order_list.jsp。

在 Struts 2 的前台 Action 配置文件 struts-front.xml 中，配置前台订单管理模块的 Action 以及视图映射关系，关键代码如下。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\action\struts-front.xml

```
<!-- 订单 Action -->
<package name="shop.order" extends="shop.front" namespace="/product">
        <action name="order_*" class="orderAction" method="{1}">
        <result name="add">/WEB-INF/pages/order/order_add.jsp</result>
        <result name="confirm">/WEB-INF/pages/order/order_confirm.jsp</result>
        <result name="list">/WEB-INF/pages/order/order_list.jsp</result>
        <result name="error">/WEB-INF/pages/order/order_error.jsp</result>
        <interceptor-ref name="customerDefaultStack"/>
        </action>
</package>
```

23.10 后台商品管理模块设计

商品是 Go 购商城的灵魂，如何管理好琳琅满目的商品信息也是 Go 购商城后台管理的一个难题，良好后台商品管理机制是一个商务网站的基石，如果没有商品信息维护，商务网站将没有意义。

23.10.1 模块概述

根据商务网站的基本要求，Go 购商城网站的商品管理模块主要实现商品信息查询、修改商品信息、删除商品信息以及添加商品信息等功能。后台商品管理模块的框架图如图 23.26 所示。

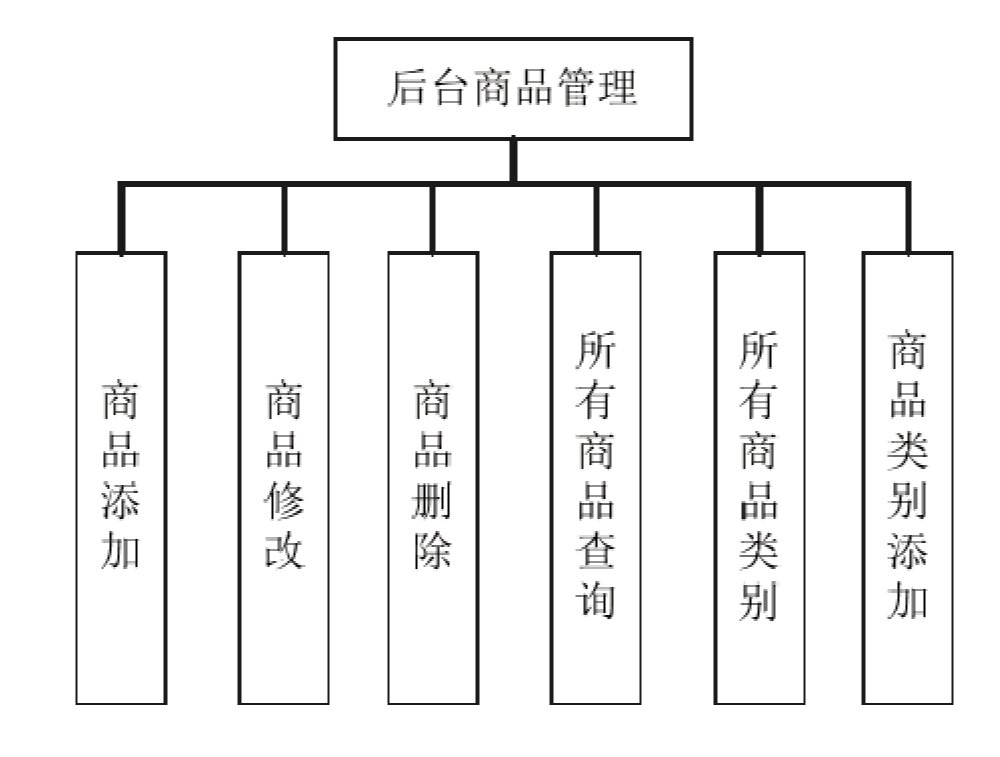


图 23.26 后台商品管理模块框架图

23.10.2　后台商品管理模块技术分析

解决 Struts2 的乱码问题可以在 struts.properties 文件中进行如下配置。

```
struts.i18n.encoding=UTF-8
```

struts.i18n.encoding 用来设置 Web 的默认编码方式，Go 购商城使用了 UTF-8 作为默认的编码方式，虽然该方法可以有效解决表单的中文乱码问题，但是该模式要求表单的 method 属性必须为 post，由于 Struts 2 中的 form 表单标签默认的 method 属性就为 post，所以不必再进行额外的设置，如果页面中的表单没有使用 Struts 2 的表单标签，需要在表单中指定 method 的属性值。

23.10.3　商品管理功能的实现

在商品管理的基本模块中，包括商品的查询、修改、删除以及添加等功能。

1. 后台商品查询

在 Go 购商城的后台管理页面中，单击左侧导航栏中的“查看所有商品”超链接，显示所有商品查询页面的运行效果如图 23.27 所示。

ID	商品名称	所属类别	采购价格	销售价格	是否推荐	适应性别	编辑	删除
1	Java编程词典	软件	98.0	120.0	true	男		
2	C#编程词典	软件	98.0	120.0	true	男		
3	.NET编程词典	软件	98.0	120.0	true	男		

首页　上一页　[1]　下一页　尾页

图 23.27　后台商品信息列表页面

后台商品列表页面实现的关键代码如下。

代码位置：光盘\TM\sl\23\Shop\WebContent\WEB-INF\pages\admin\product\ product_list.jsp

```
< table width="693" height="29" border="0" class="word01">
    <tr>
    <td width="37" height="27" align="center">ID</td>
    <td width="120" align="center">商品名称</td>
    <td width="78" align="center">所属类别</td>
    <td width="79" align="center">采购价格</td>
    <td width="79" align="center">销售价格</td>
    <td width="79" align="center">是否推荐</td>
    <td width="79" align="center">适应性别</td>
    <td width="52" align="center">编辑</td>
    <td width="52" align="center">删除</td>
```

```
        </tr>
</table>
</div>
<div id="right_mid">
<div id="tiao">
<table width="693" height="29" border="0">
        <s:iterator value="pageModel.list">
        <tr>
        <td width="37" height="27" align="center"><s:property value="id" /></td>
        <td width="120" align="center"><s:a action="product_edit" namespace="/admin/product">
        <s:param name="id" value="id"></s:param><s:property value="name" /></s:a></td>
        <td width="78" align="center"><s:property value="category.name" /></td>
        <td width="79" align="center"><s:property value="baseprice" /></td>
        <td width="79" align="center"><s:property value="sellprice" /></td>
        <td width="79" align="center"><s:property value="commend" /></td>
        <td width="79" align="center"><s:property value="sexrequest.name" /></td>
        <td width="52" align="center"><s:a action="product_edit" namespace="/admin/product">
        <s:param name="id" value="id"></s:param>
        <img src="${context_path}/css/images/rz_119.gif" width="21" height="16" /></s:a></td>
        <td width="52" align="center"><s:a action="product_del" namespace="/admin/product">
        <s:param name="id" value="id"></s:param>
        <img src="${context_path}/css/images/rz_17.gif" width="15"height="16" /></s:a></td>
        </tr>
        </s:iterator>
</table>
```

当用户单击该链接时系统将会发送一个 product_list.html 的 URL 请求，该请求执行的是 ProductAction 类中的 list()方法，ProductAction 类继承了 BaseAction 类和 ModelDriven 接口，其关键代码如下。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\action\order\ProductAction

```
public String list() throws Exception{
pageModel = productDao.find(pageNo, pageSize);                //调用公共的查询方法
        return LIST;                                          //返回后台商品列表页面
}
```

当用户单击列表中的商品名称超链接或是列表中的图标时，将进入商品信息的编辑页面，如图23.28所示。在该页面可以对商品的信息进行修改，该操作触发的是商品详细信息的查找方法，ProductAction 类中的 edit()方法，该方法将以商品的 ID 值作为查询条件，其关键代码如下。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\action\order\ProductAction

```
public String edit() throws Exception{
        this.product = productDao.get(product.getId());       //执行封装的查询方法
        createCategoryTree();                                 //生成商品的类别树
        return EDIT;                                          //返回商品信息编辑页面
}
```



图 23.28　商品信息编辑页面

商品信息编辑页面与商品添加页面的实现代码是基本相同的，区别是在编辑页面中需要显示查询到的商品信息，商品编辑页面的关键代码如下。

代码位置：光盘\TM\sl\23\Shop\WebContent\WEB-INF\pages\admin\product\product_edit.jsp

```
<table width="685" height="24" border="0">
    <tr>
    <td width="119" height="22" bgcolor="#c6e8ff" align="right">商品名称：</td>
    <td width="119" height="22" bgcolor="#c6e8ff" align="right">
    <s:textfield name="name"></s:textfield></td>
    <td rowspan="7">
    <img width="270" height="180" border="1" src="<s:property
    value="#request.get('javax.servlet.forward.context_path')"/>
    /upload/<s:property value="uploadFile.path"/>"></td>
    </tr>
    <tr>
    <td width="119" height="22" bgcolor="#c6e8ff" align="right">选择类别：</td>
    <td colspan="2">
    <s:select name="category.id" list="map" value="category.id"></s:select></td>
    </tr>
    <tr>
    <td width="119" height="22" bgcolor="#c6e8ff" align="right">采购价格：</td>
    <td colspan="2"><s:textfield name="baseprice"></s:textfield></td>
    </tr>
    <tr>
    <td width="119" height="22" bgcolor="#c6e8ff" align="right">市场价格：</td>
    <td colspan="2"><s:textfield name="marketprice"></s:textfield></td>
    </tr>
    <tr>
    <td width="119" height="22" bgcolor="#c6e8ff" align="right">销售价格：</td>
    <td colspan="2"><s:textfield name="sellprice"></s:textfield></td>
    </tr>
```

```
        <tr>
        <td width="119" height="22" bgcolor="#c6e8ff" align="right">是否为推荐：</td>
        <td colspan="2">
        <s:radio name="commend" list="#{'true':'是','false':'否'}" value="commend"></s:radio></td>
        </tr>
        <tr>
        <td width="119" height="22" bgcolor="#c6e8ff" align="right">所属性别：</td>
        <td colspan="2">
        <s:select name="sexrequest" list="@com.lyq.model.Sex@getValues()"
        value="sexrequest.getName()"></s:select></td>
        </tr>
        <tr>
        <td width="119" height="22" bgcolor="#c6e8ff" align="right">上传图片：</td>
        <td colspan="2"><s:file id="file" name="file"></s:file></td>
        </tr>
        <tr>
        <td width="119" height="22" bgcolor="#c6e8ff" align="right">商品说明：</td>
        <td colspan="2"><s:textarea name="description" cols="50" rows="6"></s:textarea></td>
        </tr>
</table>
```

2. 商品修改

当用户编辑完商品信息，单击页面中的“提交”按钮，系统将会把用户修改后的信息保存到数据库中，该操作会发送一个 product_save.html 的 URL 请求，它会调用 ProductAction 类中的 save()方法，在 save()方法中包括图片的上传和向数据表中添加数据的操作，其具体的实现代码如下。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\action\order\ProductAction

```
public String save() throws Exception{
        if(file != null ){                                          //如果文件路径不为空
        //获取服务器的绝对路径
String path = ServletActionContext.getServletContext().getRealPath("/upload");
        File dir = new File(path);
        if(!dir.exists()){                                          //如果文件夹不存在
        dir.mkdir();                                                //创建文件夹
        }
        String fileName = StringUitl.getStringTime() + ".jpg"; //自定义图片名称
        FileInputStream fis = null;                                 //输入流
        FileOutputStream fos = null;                                //输出流
        try {
        fis = new FileInputStream(file);                            //根据上传文件创建 InputStream 实例
        fos = new FileOutputStream(new File(dir,fileName)); //创建写入服务器地址的输出流对象
        byte[] bs = new byte[1024 * 4];                             //创建字节数组实例
        int len = -1;
        while((len = fis.read(bs)) != -1){                          //循环读取文件
                fos.write(bs, 0, len);                              //向指定的文件夹中写数据
                }
        UploadFile uploadFile = new UploadFile();                   //实例化对象
        uploadFile.setPath(fileName);                               //设置文件名称
```

```
        product.setUploadFile(uploadFile);                  //设置上传路径
        } catch (Exception e) {
        e.printStackTrace();
        }finally{
        fos.flush();
        fos.close();
        fis.close();
        }
        }
        //如果商品类别和商品类别 ID 不为空，则保存商品类别信息
        if(product.getCategory() != null && product.getCategory().getId() != null){
        product.setCategory(categoryDao.load(product.getCategory().getId()));
        }
        //如果上传文件和上传文件 ID 不为空，则保存文件的上传路径信息
        if(product.getUploadFile() != null && product.getUploadFile().getId() != null){
        product.setUploadFile(uploadFileDao.load(product.getUploadFile().getId()));
        }
productDao.saveOrUpdate(product);                      //保存商品信息
        return list();                                 //返回商品的查询方法
}
```

文件的上传是网络中应用最为广泛的一种技术，在 Web 应用中实现文件上传需要通过 Form 表单实现，此时表单必须以 POST 方式提交（Struts 2 标签的 form 表单默认提交方式为 POST），并且必须设置 enctype ="multipart/form-data"属性，在表单中需要实现一个或多个文件选择框供用户选择文件。当提交表单后，选择的文件内容会通过流的方式进行传递，在接收表单的 Servlet 或 JSP 页面中获取该流并将流中的数据读到一个字节数组中，此时字节数组中存储了表单请求中的内容，其中包括所有上传文件的内容，因此还需要从中分离出每个文件自己的内容，最后将分离出的这些文件写到磁盘中，完成上传操作。需要注意的是，在进行分离的过程中，操作的内容是以字节形式存在的。

3. 商品删除

当用户单击列表中的 ✖ 图标，将执行商品信息的删除操作，该操作将会向系统发送一个 product_del.html 的 URL 请求，它将触发 ProductAction 类中的 del()方法，该方法将以商品的 ID 为参数，执行持久化类中封装的 delete()方法，delete()方法中调用的是 Hibernate 的 Session 对象中的 delete()方法，其关键代码如下。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\action\order\ProductAction

```
public String del() throws Exception{
        productDao.delete(product.getId());                    //执行删除操作
        return list();                                         //返回商品列表查找方法
}
```

4. 商品添加

当用户单击后台管理页面左侧导航栏中的“商品添加”超链接时，将会进入商品添加的页面，如图 23.29 所示。

图 23.29　商品的添加页面

用户编辑完商品信息，单击页面中的“提交”按钮，该操作将会向系统发送一个 product_save.html 的 URL 请求，它与商品修改触发的是一个方法，都是 ProductAction 类中的 save()方法。

在 Struts 2 的后台 Action 配置文件 struts-admin.xml 中，配置商品管理模块的 Action 以及视图映射关系，关键代码如下。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\action\struts-admin.xml

```
<!-- 商品管理 -->
<package name="shop.admin.product" namespace="/admin/product" extends="shop.admin">
    <action name="product_*" method="{1}" class="productAction">
    <result name="list">/WEB-INF/pages/admin/product/product_list.jsp</result>
    <result name="input">/WEB-INF/pages/admin/product/product_add.jsp</result>
    <result name="edit">/WEB-INF/pages/admin/product/product_edit.jsp</result>
    <interceptor-ref name="adminDefaultStack"/>
    </action>
</package>
```

23.10.4　商品类别管理功能的实现

商品类别的维护中主要包括商品类别的查询、修改、删除以及添加。

1. 商品类别查询

商品类别在后台中分为两种，分别是商品类别树状下拉框的查询以及商品类别列表信息的查询，商品类别树状下拉框的查询的实现较为复杂一些，通过迭代的方式遍历所有的节点。

在后台的商品类别查询中，通过树状下拉框的形式展现给用户，如图 23.30 所示。

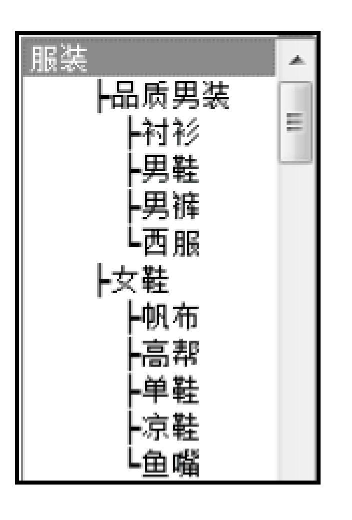


图 23.30　商品添加页面中的商品类别树状下拉框

在进入商品页面的 edit()方法中，调用了 createCategoryTree()方法用来创建商品类别树，其关键代码如下。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\action\product\ProductAction

```
private void createCategoryTree(){
String where = "where level=1";                                          //查询一级节点
     PageModel<ProductCategory> pageModel = categoryDao.find(-1, -1,where ,null);//执行查询方法
     List<ProductCategory> allCategorys = pageModel.getList();
     map = new LinkedHashMap<Integer, String>();                         //创建新的集合
     for(ProductCategory category : allCategorys){                       //遍历所有的一级节点
setNodeMap(map,category,false);                                          //将其子节点添加到集合中
     }
}
```

在 setNodeMap()方法中，首先判断节点是否为空，如果节点为空则停止遍历，程序中根据获取的节点级别为类别名称添加字符串和空格，用以生成渐进的树状结构，将拼接后的节点放入 Map 集合中，并获取其子节点重新调用 setNodeMap()方法，直到遍历的节点为空为止，其关键代码如下。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\action\product\ProductAction

```
private void setNodeMap(Map<Integer, String> map,ProductCategory node,boolean flag){
     if (node == null) {                                          //如果节点为空
     return;                                                      //返回空，结束程序运行
     }
     int level = node.getLevel();                                 //获取节点级别
     StringBuffer sb = new StringBuffer();                        //定义字符串对象
     if (level > 1) {                                             //如果不是根节点
     for (int i = 0; i < level; i++) {
     sb.append("   ");                                            //添加空格
     }
     sb.append(flag ? "├" : "└");                                 //如果为末节点添加"└"，反之添加"├"
     }
     map.put(node.getId(), sb.append(node.getName()).toString()); //将节点添加的集合中
     Set<ProductCategory> children = node.getChildren();          //获取其子节点
     // 包含子类别
     if(children != null && children.size() > 0){                 //如果节点不为空
     int i = 0;
     // 遍历子类别
     for (ProductCategory child : children) {
     boolean b = true;
     if(i == children.size()-1){                                  //如果子节点长度减 1 为 i,说明为末节点
          b = false;                                              //设置布尔常量为 false
          }
     setNodeMap(map,child,b);                                     //重新调用该方法
     }
     }
}
```

在商品添加页面中，通过<s:select>标签将商品类别树显示在下拉框中，其关键代码如下。

代码位置：光盘\TM\sl\23\Shop\WEB-INF\pages\admin\product\product_add.jsp

```
<tr>
	<td width="119" height="22" bgcolor="#c6e8ff" align="right">选择类别：</td>
	<td><s:select list="map" name="category.id"></s:select></td>
</tr>
```

当用户单击后台管理页面左侧导航栏中的“查询所有类别”超链接时，会向系统发送一个 category_list.html 的 URL 请求，它将会触发 ProductCategoryAction 类中的 list()方法，其关键代码如下。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\action\product\ProductCategoryAction

```
public String list() throws Exception{
	Object[] params = null;                                                //对象数组为空
	String where;                                                          //查询条件变量
	if(pid != null && pid > 0 ){                                           //如果有父节点
	where = "where parent.id =?";                                          //执行查询条件
	params = new Integer[]{pid};                                           //设置参数值
	}else{
	where = "where parent is null";                                        //查询根节点
	}
	pageModel = categoryDao.find(pageNo,pageSize,where,params);            //执行封装的查询方法
	return LIST;                                                           //返回后台类别列表页面
}
```

该方法将返回后台的商品类别列表页面，如图 23.31 所示。

ID	类别名称	子类别	添加子类别	所属父类	编辑	删除
1	服装	有9个子类别	添加	无		✕
51	配饰	有6个子类别	添加	无		✕
83	家居	有8个子类别	添加	无		✕

首页 上一页 [1] 下一页 尾页

图 23.31　后台商品类别信息列表

2. 商品类别添加

单击导航栏中“添加商品类别”或是商品类别列表页面中的“添加”超链接时，会进入到商品类别的添加页面，如图 23.32 所示。



图 23.32　商品类别添加页面

在“类别名称”文本框中输入类别名称后，单击“提交”按钮，将会触发 ProductCategoryAction 类中的 save()方法，在 save()方法中首先判断该节点的父节点参数是否存在，如果存在则先设置其父节



点属性，然后再保存商品类别信息，其关键代码如下。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\action\product\ProductCategoryAction

```
public String save() throws Exception{
      if(pid != null && pid > 0 ){                                    //如果有父节点
      category.setParent(categoryDao.load(pid));                      //设置其父节点
      }
      categoryDao.saveOrUpdate(category);                                 //添加类别信息
      return list();                                                      //返回类别列表的查找方法
}
```

3. 商品类别修改

当网站管理员单击商品类别列表中的按钮时，将进入商品类别修改的页面，如图 23.33 所示。



图 23.33　商品类别修改页面

修改商品类别信息完毕后，单击页面中的“提交”按钮，其触发的也是商品类别添加中 ProductCategoryAction 类的 save()方法。

4. 商品类别删除

当用户单击商品类别列表中的图标，将执行商品类别信息的删除操作，该操作将会向系统发送一个 category_del.html 的 URL 请求，它将触发 ProductCategoryAction 类中的 del()方法，该方法将以商品类别的 ID 为参数，执行持久化类中封装的 delete()方法，删除指定的信息，其关键代码如下。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\action\product\ProductCategoryAction

```
public String del() throws Exception{
      if(category.getId() != null && category.getId() > 0){        //判断是否获得 ID 参数
      categoryDao.delete(category.getId());                        //执行删除操作
      }
      return list();                                               //返回商品类别列表的查找方法
}
```

在商品类别管理中添加、修改以及删除的操作实现都较为简单，商品类别信息的查询方法支持无限级的树状分级查询。

在 Struts 2 的后台 Action 配置文件 struts-admin.xml 中，配置商品类别管理模块的 Action 以及视图映射关系，关键代码如下。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\action\struts-admin.xml

```
<!-- 类别管理 -->
<package name="shop.admin.category" namespace="/admin/product" extends="shop.admin">
      <action name="category_*" method="{1}" class="productCategoryAction">
      <result name="list">/WEB-INF/pages/admin/product/category_list.jsp</result>
```

```
        <result name="input">/WEB-INF/pages/admin/product/category_add.jsp</result>
        <result name="edit">/WEB-INF/pages/admin/product/category_edit.jsp</result>
        <interceptor-ref name="adminDefaultStack"/>
        </action>
</package>
```

23.11　后台订单管理模块的设计

网站管理员可以对会员的订单进行维护，但这种维护只能修改订单的状态，并不能修改订单中的任何信息，因为当用户确认订单时该订单就已经生效，它相当于用户与商家交易的一个契约，是用户与商家之间的一个交易凭证，所以不能进行任何的修改。

23.11.1　模块概述

在后台的订单管理模块中，主要分为两个基本模块，分别是订单的查询和订单状态的修改，其中订单的查询又可分为订单的全部查询和用户自定义的条件查询，框架模块图如图 23.34 所示。

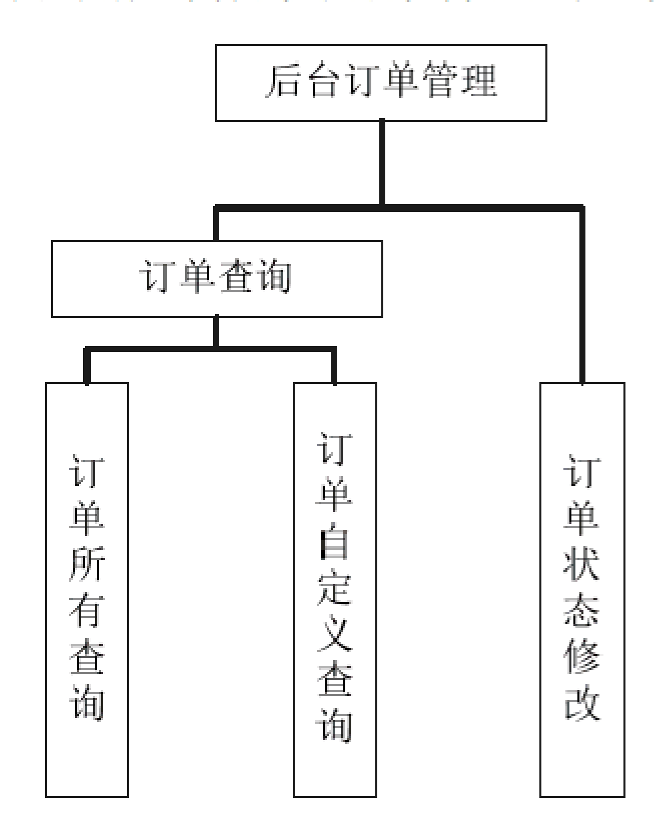


图 23.34　订单管理模块框架图

23.11.2　后台订单管理模块技术分析

当用户在订单列表页面中，单击“更新订单状态”按钮时，将会弹出提示对话框，让用户选择修改的状态信息，在订单列表页面中通过模态窗体的形式弹出该对话框，为“更新”按钮绑定触发事件的关键代码如下。

代码位置：光盘\TM\sl\23\Shop\WebContent\pages\admin\order\order_list.jsp

```
<td width="150" align="center">
```



```
<s:url action="order_select" namespace="/admin/product" var="order_select">
        <s:param name="orderId" value="orderId"></s:param></s:url>
<input type="button" value="更新订单状态"onclick="openWindow('${order_select}',350,150);">
</td>
```

在弹出的子窗体中，如果是模态的，子窗体不关闭将无法进行主窗体的任何操作；如果是非模态的，子窗体不关闭同样可以进行主窗体的操作。

根据页面中的代码可知，Action 请求 order_select 跳转的页面为 order_select.jsp，也就是弹出的模态窗体。

当用户在弹出的模态窗体中单击该页面，将会向系统发送一个 order_update.html 的 URL 请求，该请求触发的是 OrderAction 类中的 update()方法，其关键代码如下。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\action\order\OrderAction

```
public String update() throws Exception {
        OrderState orderState = order.getOrderState();                    //获取设置的订单状态
        order = orderDao.load(order.getOrderId());                        //装载订单对象
        order.setOrderState(orderState);                                  //设置的订单状态
        orderDao.update(order);                                           //修改订单状态
        return "update";                                                  //放回订单状态修改成功页面
}
```

当订单状态修改成功后，会弹出订单信息修改成功的提示对话框，通过 JavaScript 对该窗体进行设置，该窗体 3 秒后自动关闭，并刷新主页面。

在订单更新成功的页面中，设置窗体自动关闭的 JavaScript 关键代码如下。

代码位置：光盘\TM\sl\23\Shop\WebContent\pages\admin\order\ order_update_success.jsp

```
<script type="text/javascript">
        function closewindow(){
        if(window.opener){
        window.opener.location.reload(true);                              //刷新父窗体
        window.close();                                                   //关闭提示窗体
        }
}
function clock(){
        i = i -1;
        if(i > 0){                                                        //如果 i 大于 0
        setTimeout("clock();",1000);                                      //1 秒后重新调用 clock()方法
        }else{
        closewindow();                                                    //调用关闭窗体方法
        }
}
        var i = 3;                                                        //设置 i 值
        clock();                                                          //页面加载后自动调用 clock()
</script>
```

在上述程序中通过变量 i 来设置窗体自动关闭的时间，在 clock()方法中，当 i 的值为零时调用关闭窗体的方法，并且通过 setTimeout()方法设置方法调用时间，参数“1000”的单位为毫秒。

23.11.3　后台订单查询的实现

在后台订单的查询中，分为所有订单查询和订单的自定义条件查询，在订单的自定义查询中用户可以根据设定不同的查询条件查询订单的指定信息。在本应用程序中两者调用的都是同一个查询方法。

在管理页面左侧导航栏中单击“查看订单”超链接时，将进入订单状态管理页面，如图 23.35 所示。

订单号	总金额	消费者	支付方式	创建时间	订单状态	修改
201005041012220323561	120.0	mrsoft	邮局汇款	2010年05月4日 10:12	已发货	更新订单状态
201004271843190764180	240.0	mrsoft	邮局汇款	2010年04月27日 18:43	已发货	更新订单状态
201004260941250469034	120.0	mrsoft	邮局汇款	2010年04月26日 09:41	已发货	更新订单状态

首页　上一页　[1]　下一页　尾页

图 23.35　订单状态管理页面

当用户单击左侧导航栏中的“订单查询”超链接时，将会进入订单查询条件的自定义页面，如图 23.36 所示。

订单查询

订单号：

订单状态：已发货　交易完成　已取消

会员名：

收款人：

提 交

图 23.36　订单查询条件的自定义页面

当用户单击导航栏中的“查看订单”超链接或单击订单查询条件的自定义页面中的“提交”按钮时，都会向系统发送一个 order_list.html 的 URL 请求，该请求触发的是 OrderAction 中的 list()方法，其关键代码如下。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\action\order\OrderAction

```
public String list() throws Exception {
    Map<String, String> orderby = new HashMap<String, String>(1);    //定义 Map 集合
    orderby.put("createTime", "desc");                               //设置按创建时间倒序排列
    StringBuffer whereBuffer = new StringBuffer("");                 //创建字符串对象
    List<Object> params = new ArrayList<Object>();
    if(order.getOrderId() != null && order.getOrderId().length() > 0){//如果订单号不为空
    whereBuffer.append("orderId = ?");                               //以订单号为查询条件
    params.add(order.getOrderId());                                  //设置参数
    }
    if(order.getOrderState() != null){                               //如果订单状态不为空
    if(params.size() > 0) whereBuffer.append(" and ");               //增加查询条件
    whereBuffer.append("orderState = ?");                            //设置订单状态为查询条件
```

```
        params.add(order.getOrderState());                          //设置参数
        }
        if(order.getCustomer() != null && order.getCustomer().getUsername() != null
        && order.getCustomer().getUsername().length() > 0){          //如果会员名不为空
        if(params.size() > 0) whereBuffer.append(" and ");           //增加查询条件
        whereBuffer.append("customer.username = ?");                 //设置会员名为查询条件
        params.add(order.getCustomer().getUsername());               //设置参数
        }
        if(order.getName() != null && order.getName().length()>0){   //如果收款人姓名不为空
        if(params.size() > 0) whereBuffer.append(" and ");           //增加查询条件
        whereBuffer.append("name = ?");                              //设置收款人姓名为查询条件
        params.add(order.getName());                                 //设置参数
        }
        //如果 whereBuffer 为空则查询条件为空，否则以 whereBuffer 为查询条件
        String where = whereBuffer.length()>0 ? "where "+whereBuffer.toString() : "";
        pageModel = orderDao.find(where, params.toArray(), orderby, pageNo, pageSize);  //执行查询方法
        return LIST;                                                               //返回后台订单列表
}
```

“查看订单”超链接并没有为 list()方法传递任何的参数，所以最后传给 find()方法的 where 查询条件字符串为空，find()方法将会从数据库中查询所有的订单信息并按创建时间的倒序输出。

list()方法将会返回后台的订单信息列表页面，在该页面利用 Struts 2 的<s:iterator>方法遍历输出返回结果集中的信息即可。后台订单信息列表页面的关键代码如下。

代码位置：光盘\TM\sl\23\Shop\WebContent\pages\admin\order\order_list.jsp

```
<table width="693" height="29" border="0" class="word01">
        <tr>
        <td width="140" align="center">订单号</td>
        <td width="60" align="center">总金额</td>
        <td width="63" align="center">消费者</td>
        <td width="70" align="center">支付方式</td>
        <td width="140" align="center">创建时间</td>
        <td width="70" align="center">订单状态</td>
        <td width="150" align="center">修改</td>
        </tr>
</ table >
<table width="693" height="29" border="0">
        <s:iterator value="pageModel.list">
        <tr>
        <td width="140" align="center"><s:property value="orderId" /></td>
        <td width="60" align="center"><s:property value="totalPrice" /></td>
        <td width="63" align="center"><s:property value="customer.username" /></td>
        <td width="70" align="center"><s:property value="paymentWay.getName()" /></td>
        <td width="140" align="center">
//遍历操作
         <s:date name="createTime" format="yyyy 年 MM 月 d 日  HH:mm" /></td>
        <td width="70" align="center"><s:property value="orderState.getName()" /></td>
        <td width="150" align="center">
        <s:url action="order_select" namespace="/admin/product" ar="order_select">
```

```
        <s:param name="orderId" value="orderId"></s:param>
        </s:url> <input type="button" value="更新订单状态"
onclick="openWindow('${order_select}',350,150);"></td>
        </tr>
        </s:iterator>
</table>
```

23.12　开发技巧与难点分析

23.12.1　解决订单号为空时查询报错

在代码初期对后台订单自定义查询进行测试的时候发现，如果订单号为空，填写后面的查询条件进行查询的时候就会报错，如图 23.37 所示。

```
Struts Problem Report

Struts has detected an unhandled exception:

            1. unexpected token: and near line 1, column 39 [from
               com.lyq.model.order.Order where and orderState = ? order by
               createTime desc]
            2. unexpected token: and near line 1, column 39 [from
               com.lyq.model.order.Order where and orderState = ? order by
Messages:      createTime desc]; nested exception is
               org.hibernate.hql.ast.QuerySyntaxException: unexpected token: and
               near line 1, column 39 [from com.lyq.model.order.Order where and
               orderState = ? order by createTime desc]
```

图 23.37　订单号为空查询时系统的报错信息

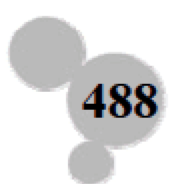
从错误信息中可以发现在 where 条件查询语句中关键字 where 直接连接的是关键字 and，也就是说在订单状态前置查询条件订单号码为空的情况下同样将关键字 and 拼接到了查询条件中，解决该问题只需对查询条件的 Object 数组的长度进行判断，如果数组的长度不为 0，就将关键字 and 添加到查询条件中，以订单状态的查询条件为例，其关键代码如下。

代码位置：光盘\TM\sl\23\Shop\src\com\lyq\action\order\OrderAction

```
if(order.getOrderState() != null){                //如果订单状态不为空
        if(params.size() > 0){
        whereBuffer.append(" and ");               //增加查询条件
        }
        …                                          //省略的代码
}
```

23.12.2　通过 Struts 2 的拦截器来解决 Session 超时出现空指针异常的问题

从网站的安全性考虑，用户在没有登录或是 Session 失效的时候是不允许进行购物和后台维护的，



一般客户端的 Session 是有时间限制的（根据服务器中的配置决定，一般为 20 分钟），如果超出时间限制，系统就会报出现空指针的异常信息，出现这种情况的原因是系统从 Session 中取得的信息为空，即获取的用户登录信息为空，空值造成了这种情况的发生。

这个问题可以通过 Struts 2 的拦截器来解决，根据拦截器判断 Session 是否为空，并根据判断结果执行不同的操作。

拦截器（Interceptor）是 Struts 2 框架中一个非常重要的核心对象，它可以动态增强 Action 对象的功能，通过对登录拦截器的配置，如果会员的 Session 失效后，用户将无法使用购物车功能，除非重新登录；如果管理员 Session 失效后，将无法进入后台进行操作，没有直接登录的用户在地址栏中直接输入 URL 地址也将被拦截器拦截，并返回系统的登录页面，这样很大程度地提升了系统的安全性。

拦截器动态地作用于 Action 与 Result 之间，它可以动态地对 Action 以及 Result 进行增强（在 Action 与 Result 中加入新功能）。当客户端发送请求时，会被 Struts 2 的过滤器所拦截，此时 Struts 2 对请求持有控制权，Struts 2 会创建 Action 的代理对象，并通过一系列的拦截器对请求进行处理，最后再交给指定的 Action 进行处理。拦截器实现的核心思想是 AOP（Aspect Oriented Progamming，面向切面编程）。

首先创建会员登录拦截器 CustomerLoginInteceptor，其关键代码如下。

```
public String intercept(ActionInvocation invocation) throws Exception {
      ActionContext context = invocation.getInvocationContext();            //获取 ActionContext
      Map<String, Object> session = context.getSession();                  //获取 Map 类型的 session
      if(session.get("customer") != null){                                 //判断用户是否登录
      return invocation.invoke();                                          //调用执行方法
      }
      return BaseAction.CUSTOMER_LOGIN;                                    //返回登录
}
```

在前台的 Struts 2 的配置文件中配置该拦截器，其关键代码如下。

```
<package name="shop.front" extends="shop-default">
<!-- 配置拦截器 -->
<interceptors>
      <!-- 验证用户登录的拦截器 -->
      <interceptor name="loginInterceptor" class="com.lyq.action.interceptor.CustomerLoginInteceptor"/>
      <interceptor-stack name="customerDefaultStack">
            <interceptor-ref name="loginInterceptor"/>
            <interceptor-ref name="defaultStack"/>
            </interceptor-stack>
      </interceptors>
      <action name="index" class="indexAction">
            <result>/WEB-INF/pages/index.jsp</result>
      </action>
</package>
```

通常情况下，拦截器对象实现的功能比较单一，它类似于 Action 对象的一个插件，为 Action 对象动态地织入新的功能。

系统后台拦截器的配置与前台类似，这里不再进行详细的说明。

23.13 小　　结

本系统只是实现了电子商务网站一些基本的功能，真正的商务网站的开发难度和工作量要比本系统复杂和烦琐得多，但是希望通过本系统的开发，读者可以了解网站开发的简单流程、SSH2 框架的整合以及 MVC 的设计模式，相信读者可以融会贯通。